Entfernte Redundanz

Beschreibung, Analyse und Bewertung eines neuartigen Architekturkonzeptes für fehlertolerante Steuerungssysteme

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

durch die Fakultät für Wirtschaftswissenschaften der
Universität Duisburg-Essen

vorgelegt von

Dipl.-Wirt.Inf. Thorsten Kimmeskamp

aus Essen

Essen, 28. September 2011

Tag der mündlichen Prüfung: 19. Dezember 2011
Erstgutachter: Prof. Dr. Klaus Echtle
Zweitgutachter: Prof. Dr. Bruno Müller-Clostermann

Abstract

Die Frage, wie sicherheitskritische Steuerungen fehlertolerant gestaltet werden können, kann als grundsätzlich beantwortet angesehen werden (Echtle, 1990). Teils seit Jahrzehnten existieren Verfahren zur Fehlererkennung, Wiederholung misslungener Rechenoperationen oder paralleler Programmausführung auf mehreren Rechnern. Auf ein Gesamtsystem, wie z. B. ein Automobil bezogen, wurde jedoch bislang meist jedes einzelne Teilsystem (Lenkanlage, Bremsen etc.) isoliert betrachtet, was im Ergebnis zu massiver, aus Fehlertoleranzsicht aber entbehrlicher, struktureller Redundanz führte.

Aus dieser Überlegung heraus entstand das Konzept der „Entfernten Redundanz“, welches es erlaubt, im Gesamtsystem vorhandene Ressourcen unabhängig vom Ort ihres Vorhandenseins nutzbar zu machen. Als Folge kann der Redundanzgrad auf das zur fehlertoleranten Ansteuerung der Peripherie unbedingt notwendige Maß beschränkt werden. Zu diesem Zweck werden Sensoren und Aktuatoren nur an bestimmte, z. B. räumlich nahegelegene Steuergeräte unmittelbar angeschlossen, während die zugehörige Programmlogik auf beliebigen Knoten im Netzwerk ablaufen kann. Der Datenaustausch zwischen diesen „entfernten“ Komponenten findet signaturgeschützt mittels eines in einem solchen System i. d. R. bereits existierenden Bussystems über fremde Rechner hinweg statt. Gegenseitige Kontrolle garantiert dabei zu herkömmlich ausgeführten Systemen (im Rahmen dieser Arbeit als „Dedizierte Redundanz“ bezeichnet) identische Fehlertoleranzeigenschaften. Bei Nutzung Entfernter Redundanz jedoch können Hardwarekomponenten durch auf einem fremden Teilsystem ausgeführte Software ersetzt werden, was im Hinblick auf die für das Gesamtsystem entstehenden Kosten ein wesentliches Einsparpotenzial darstellt. Aber auch in Bezug auf ein einzelnes Teilsystem ermöglicht es der Einsatz Entfernter Redundanz, aufwändige (und darüber hinaus fehleranfällige) Verkabelungsstrukturen in beträchtlichem Umfang zu reduzieren. Alleine die hierdurch entstehende Gewichtsreduktion kann je nach Anwendungsgebiet (so z. B. im Flugzeugbau) von erheblicher Bedeutung sein.

Die durch Entfernte Redundanz entstehende Systemarchitektur ist dabei nahezu frei skalierbar und nicht etwa auf einen bestimmten, typischerweise durch die Anwendung vorgegebenen Fehlertoleranzgrad eingeschränkt. Realisierbar sind dementsprechend neben den beiden in der Praxis häufigsten Anforderungen − Ausfallsicherheit und Einfehlertoleranz − beliebige n-von-m-Systeme.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AFDX: Avionics Full Duplex Switched Ethernet

ASIC: Application-Specific Integrated Circuit

ASIL: Automotive Safety Integrity Level

AUTOSAR: Automotive Open System Architecture

CAN: Controller Area Network

CRC: Cyclic Redundancy Check

CTL: Computation Tree Logic

EASIS: Electronic Architecture and System Engineering for Integrated Safety Systems

ECU: Electronic Control Unit

EDA: Electronic Design Automation

EDIF: Electronic Design Interchange Format

FPGA: Field Programmable Gate Array

IMA: Integrated Modular Avionics

MMU: Memory Management Unit

MTTF: Mean Time To Failure

POST: Power-On Self-Test

RAID: Redundant Array of Independent Disks

SIL: Safety Integrity Level

SPI: Serial Peripheral Interface

TCN: Train Communication Network

TMR: Triple Modular Redundancy

VHDL: Very High Speed Integrated Circuit Hardware Description Language

WCET: Worst Case Execution Time

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation, Zielsetzung und Aufbau der Arbeit

Fehlertolerante Systeme sollen die Funktion des Gesamtsystems auch beim Auftreten bestimmter Fehler in einzelnen Komponenten oder Teilsystemen gewährleisten. Der Grund dafür, dass eine solche Anforderung gestellt wird, liegt typischerweise darin, dass ein Komplettausfall einen hohen finanziellen Schaden oder gar eine Gefahr für Leib und Leben verursachen würde, Fehler in Bezug auf einzelne Komponenten jedoch nicht hinreichend unwahrscheinlich sind. Sofern diese Wahrscheinlichkeit nicht (oder nicht wirtschaftlich) durch außergewöhnliche Sorgfalt beim Fertigungsprozess oder Verwendung besonders hochwertiger Materialien reduziert werden kann, bleibt als einziger Ausweg der Einsatz von Redundanz, d. h. von Mitteln, die für den eigentlichen Nutzbetrieb entbehrlich wären, im Fehlerfall jedoch z. B. als Ersatz für defekte Komponenten dienen oder auf übergeordneter Ebene Fehlertoleranzverfahren koordinieren. Wenngleich auch hierdurch niemals eine absolute Schadensfreiheit garantiert werden kann, so lässt sich dennoch die Zuverlässigkeit des Gesamtsystems beträchtlich über die seiner Komponenten hinaus steigern.

Das Prinzip der Fehlertoleranz ist grundsätzlich nicht auf die Informationstechnik beschränkt, seine Bedeutung zeigt sich aber besonders deutlich in den seit vielen Jahren von ihr in zunehmendem Maße durchdrungenen ganz alltäglichen Anwendungsgebieten, wie etwa Verkehrssystemen, Telekommunikation, Energiewirtschaft, Medizintechnik oder Bankwesen. Nicht behandelte Fehler können dort beispielsweise dazu führen, dass

- Piloten durch falsche Anzeigen zu gefährlichen Manövern veranlasst werden,
- Fahrerassistenzsysteme durch ein Eingreifen in die Lenkung Unfälle verursachen,
- der Handel an Wertpapierbörsen gestört oder verhindert wird oder
- Havarien in Kernkraftwerken zu unkontrollierbaren Katastrophen führen.

Das grundlegende Problem, Rechensysteme fehlertolerant zu gestalten, kann bereits seit Längerem als gelöst betrachtet werden (Echtle, 1990). Vor dem Hintergrund eines zunehmenden Kostendrucks, aber auch angesichts der steigenden Verbreitung eingebetteter Systeme mit besonders knapp bemessenen Ressourcen ist es jedoch weiterhin Gegenstand der Forschung, möglichst effiziente Fehlertoleranzverfahren zu entwickeln. Auch die vorliegende Arbeit widmet sich, bezogen auf ein bestimmtes Anwendungsfeld, diesem Thema.

Gegenstand dieser Arbeit sind **fehlertolerante Steuerungen**. Ihre Besonderheit (und damit zugleich der Ausgangspunkt für mögliche Optimierungen) liegt in der kontinuierlichen Interaktion mit der sie umgebenden Umwelt mit Hilfe von Sensoren und Aktuatoren. Bereits an ein einzelnes Steuergerät in einem aktuellen Kraftfahrzeug kann eine zweistellige Zahl an Sensoren und Aktuatoren angeschlossen sein. Insgesamt sind, für den Nutzer größtenteils im Verborgenen, möglicherweise über 50 Steuergeräte im Einsatz (EASIS, 2005). Wenn nun noch einige Teilsysteme redundant ausgelegt werden – und dies ist i. d. R. erforderlich, wenn die Sicherheit der Fahrzeuginsassen von ihrer fehlerfreien Funktion abhängt – sind die Folgen für die Komplexität und die Kosten des Gesamtsystems offensichtlich.

Die Grundidee des in dieser Arbeit vorgestellten Konzepts der **Entfernten Redundanz** besteht nun darin, die Voraussetzungen dafür zu schaffen, dass die einzelnen Komponenten eines fehlertoleranten Systems nicht mehr zwingend unmittelbar miteinander verbunden sein müssen. Stattdessen soll der Zugriff auf redundante Hardware (d. h. Rechner, Sensoren und Aktuatoren) über fremde Steuergeräte hinweg ermöglicht werden, ohne dabei die Fehlertoleranzeigenschaften zu beeinträchtigen. Der erhoffte Nutzen eines solchen Verfahrens lässt sich wie folgt skizzieren:

- Der Systemaufbau kann stark vereinfacht werden, da aufwändige Verkabelungsstrukturen in erheblichem Umfang entfallen. Als Ersatz dient die signaturgeschützte Kommunikation über typischerweise bereits vorhandene Bussysteme.
- Durch den Wegfall der Notwendigkeit einer direkten Verkabelung mit der Peripherie können Steuergeräte durch auf einem fremden Rechner ausgeführte Software ersetzt werden. Freie Kapazität vorausgesetzt, lässt sich so die Gesamtzahl der Steuergeräte reduzieren.
- Da Funktion und Ort der Funktionserbringung gedanklich voneinander getrennt werden, lassen sich typische „Entwurfsmuster“ fehlertoleranter Systeme besser erkennen und neue Möglichkeiten, diese umzusetzen, werden sichtbar.

Während bereits existierende Integrationsbemühungen wie AUTOSAR[1] im Automobilbau oder IMA[2] im Luftfahrtbereich als eher technologiebezogen charakterisiert werden können, liegt das besondere Augenmerk dieser Arbeit auf den an ein gegebenes

[1] Automotive Open System Architecture
[2] Integrated Modular Avionics

System gestellten Fehlertoleranzanforderungen. Die Details der technischen Umsetzung können folglich in vielen Punkten generisch bleiben und sind nicht von einer bestimmten Hardware-/Softwarearchitektur abhängig.

Eine erste Zielsetzung der vorliegenden Arbeit ist die detaillierte Beschreibung des Konzepts der Entfernten Redundanz und seiner Anwendung für verschiedene Grade von Fehlertoleranzanforderungen sowie im allgemeinen Fall. Während dabei die grundlegenden Merkmale des Verfahrens im Vordergrund stehen, soll im weiteren Verlauf anhand eines durchgehenden Fallbeispiels der Nachweis seiner Eignung für den industriellen Einsatz erbracht werden. Vor diesem Hintergrund wird abschließend eine Bewertung des Konzepts hinsichtlich seines Beitrags für die Fehlertoleranzforschung sowie in Bezug auf verschiedene die Praxis berührende Fragestellungen vorgenommen.

Hierzu werden zunächst in Kapitel 2 die notwendigen Methoden für die Formulierung von Fehlertoleranzanforderungen vorgestellt. In Kapitel 3 wird sodann die Rolle der Redundanz als Grundlage der Fehlertoleranz und Ausgangspunkt der vorliegenden Arbeit dargelegt. Im darauffolgenden Kapitel 4 wird das Konzept der Entfernten Redundanz beschrieben und in Bezug zum (in dieser Arbeit als „Dedizierte Redundanz“ bezeichneten) gegenwärtigen Stand der Technik gesetzt. In diesem Rahmen werden neben der Grundidee des Konzepts die Voraussetzungen seiner Anwendbarkeit, seine Ausgestaltung für unterschiedliche Fehlertoleranzanforderungen sowie grundsätzlich bestehende Freiheitsgrade und die sich daraus ergebenden Möglichkeiten für Optimierungen in Bezug auf das Verfahren selbst aufgezeigt. Kapitel 5 beinhaltet die umfassende Analyse eines Beispielsystems, in diesem Fall einer auf dem Konzept der Entfernten Redundanz basierenden, fehlertoleranten, elektronisch geregelten Lenkung. Unter Verwendung verschiedener Modellwelten und Detaillierungsgrade (Fehlerbäume, formale Verifikation, funktionale Simulation, Hardware-Demonstrator) werden unterschiedliche Aspekte des Verfahrens betrachtet und zugleich seine Verwendbarkeit für industrierelevante Anwendungen belegt. Kapitel 6 nimmt eine Bewertung des Ansatzes vor. Diese erfolgt vor dem Hintergrund der Fehlertoleranzforschung, nimmt aber auch Bezug auf ökonomische Faktoren, technische Entwicklungen, die Auswirkungen auf den Entwicklungsprozess und rechtliche Rahmenbedingungen. In Kapitel 7 folgt schließlich eine Zusammenfassung der Ergebnisse und ein Ausblick auf sich aus der Arbeit ergebende, weiterführende Forschungsfragen.

2 Fehlertoleranzanforderungen und ihre Spezifikation

Fehler können die Verlässlichkeit eines Systems hinsichtlich einer Vielzahl unterschiedlicher Eigenschaften beeinträchtigen. (Avižienis, Laprie, Randell, & Landwehr, 2004) führen diesbezüglich Verfügbarkeit, Zuverlässigkeit, Sicherheit, Integrität und Wartbarkeit auf, weisen jedoch darauf hin, dass es vom konkreten Einzelfall abhängt, welcher dieser Aspekte bei der Betrachtung im Vordergrund stehen muss. (Echtle, 1990) unterscheidet allgemeiner:

- **Zuverlässigkeit** als Maß für die Fähigkeit eines Systems, seine spezifizierte Funktion unter zulässigen Bedingungen zu erbringen und
- **Sicherheit** als die Abwesenheit von Gefahr, d. h. eines Zustandes, in dem unter anzunehmenden Bedingungen ein Schaden eintreten kann.

Es sei in diesem Zusammenhang erwähnt, dass beide Systemeigenschaften nicht notwendigerweise zusammenfallen müssen: Ein Flugzeug, welches etwa aufgrund festgestellter Mängel stets am Boden bleibt, ist zwar sicher, aber im Sinne der Definition nicht zuverlässig. Einige Fehlertoleranzverfahren sind sogar explizit darauf ausgelegt, einen sicheren Zustand zu erreichen, ohne weiterhin Zuverlässigkeit zu gewährleisten. Im umgekehrten Fall ist es jedoch meist so, dass die Spezifikation gewisse Sicherheitsanforderungen beinhaltet, so dass aus der Zuverlässigkeit bereits die Sicherheit folgt. Insoweit dies jedoch nicht gefordert wird, ist zumindest theoretisch auch Zuverlässigkeit ohne Sicherheit möglich.

Die oben genannten Eigenschaften erlauben es nicht nur, die Anforderungen an ein System zu spezifizieren, sondern auch ihre tatsächliche Erreichung zu messen. Hierzu werden nun zur späteren Verwendung einige Kenngrößen eingeführt. Bezogen auf die Eigenschaft der *Zuverlässigkeit* definiert (Echtle, 1990):

Die **Lebensdauer $\boldsymbol{L}$** eines nichtreparierbaren Systems ist eine reellwertige Zufallsvariable, welche die Zeitdauer von der Inbetriebnahme des Systems bis zu seinem Ausfall angibt. Die Wahrscheinlichkeitsdichte der Lebensdauer in Abhängigkeit von der Zeit sei mit $f(t)$ bezeichnet. Die zugehörige Verteilungsfunktion gibt als **Fehlerwahrscheinlichkeit $\boldsymbol{F(t)}$**

an, mit welcher Wahrscheinlichkeit ein bei Inbetriebnahme fehlerfreies System innerhalb des Zeitintervalls $[0, t]$ fehlerhaft wird. Es gilt:

$$F(t) = \int_0^t f(t)dt$$

mit $F(0) = 0$ und $\lim_{t\to\infty} F(t) = 1$.

Das Komplement ist die **Überlebenswahrscheinlichkeit $\boldsymbol{R(t)}$.** Sie gibt die Wahrscheinlichkeit an, dass ein zu Beginn fehlerfreies System bis zum Zeitpunkt t ohne Unterbrechung fehlerfrei bleibt. Es gilt daher:

$$R(t) = 1 - F(t)\epsilon[0,1]$$

mit $R(0) = 1$ und $\lim_{t\to\infty} R(t) = 0$.

Die **Mittlere Lebensdauer $\boldsymbol{E(L)}$** eines Systems gibt den Erwartungswert der Zeitdauer bis zum Ausfall an. Es ist:

$$E(L) = \int_{-\infty}^{\infty} t \cdot f(t)\, dt = \int_0^{\infty} R(t)dt$$

Anmerkung: Im Englischen ist hierfür die Bezeichnung „Mean Time To Failure“, $MTTF$ gebräuchlich; für eine Herleitung der letzten Identität vgl. (Echtle, 2008).

Die **Ausfallrate $\boldsymbol{z(t)}$** ist der Anteil der zu einem bestimmten Zeitpunkt ausfallenden Komponenten bezogen auf die Gesamtzahl der zu diesem Zeitpunkt noch fehlerfreien Komponenten:

$$z(t) = \frac{f(t)}{R(t)}$$

Oftmals wird eine exponentialverteilte Lebensdauer mit einer über die Zeit konstanten Ausfallrate $z(t) = \lambda$ angenommen, so dass sich die übrigen Kenngrößen zu $F(t) = 1 - e^{-\lambda \cdot t}$, $R(t) = e^{-\lambda \cdot t}$ und $E(L) = \frac{1}{\lambda}$ ergeben.

Völlig analog lassen sich entsprechende Kenngrößen bezogen auf die Systemeigenschaft der *Sicherheit* definieren. Da diese im Rahmen der vorliegenden Arbeit jedoch nicht benötigt werden, sei hierfür auf o. g. Quelle verwiesen.

Der Begriff des **Fehlers** selbst ist im Deutschen zumindest potenziell unscharf. Im Englischen ist eine genauere Trennung möglich, so wird, etwa bei (Laprie, 1985), zwischen *failure* als dem nach außen hin (in Form von Nichterfüllung der Spezifikation) beobachtbaren Funktionsausfall, *error* als dem zugrundeliegenden inneren Fehlzustand des betroffenen Systems und *fault* als der Ursache hierfür unterschieden. Da es in der Regel vom Standpunkt des Betrachters abhängt, welcher der genannten Aspekte eines Gesamtereignisses als relevant erachtet wird, erscheint jedoch die Verwendung des allgemeineren Begriffs des *Fehlers* gerechtfertigt, sofern aus dem Kontext hervorgeht, was gemeint ist. Geht man zudem davon aus, dass eine vollständige Spezifikation des betrachteten Systems vorliegt, so lässt sich in allgemeiner Form sagen:

Fehlertoleranz beschreibt „die Fähigkeit eines Systems, auch mit einer begrenzten Anzahl fehlerhafter Komponenten seine spezifizierte Funktion zu erfüllen“ (Echtle, 1990, S. 9).

Aufgrund der Tatsache, dass die Anforderung der Fehlertoleranz nicht in beliebiger Weise, sondern nur in Bezug auf eine konkrete **Fehlervorgabe**, also eine Aufstellung der zu tolerierenden Fehler, erfüllt werden kann, ist es notwendig, zunächst in einem **Fehlermodell** anzugeben, welche Fehler grundsätzlich als möglich betrachtet werden. Da hierfür sowohl die von einem Fehler betroffenen Komponenten, als auch die Art ihrer Fehlfunktion anzugeben sind, ist ebenfalls eine geeignete **Systembeschreibung** erforderlich. Ein Ansatz für ein mathematisches Modell, welches alle drei genannten Bereiche abdeckt, wird in (Kimmeskamp, 2005) dargestellt.

Im Rahmen dieser Arbeit werden (je nach betrachtetem Teilaspekt des Konzepts der Entfernten Redundanz) Systembeschreibungen von unterschiedlichem Abstraktionsgrad verwendet. Aus diesem Grund sind auch die Fehlermodelle in Anpassung daran unterschiedlich detailliert. So wird bei allgemeineren Modellen zunächst lediglich die Anzahl der betroffenen Komponenten betrachtet (sog. k-Fehler-Annahme), ohne dass die Art der Fehlfunktion näher definiert wird. Sobald jedoch bei Betrachtung der genauen Funktionalität unterschiedliche

Fehler (wie etwa ein falscher, aber syntaktisch gültiger Wert im Gegensatz zu einem als ungültig erkennbaren Wert) unterschiedliche Gegenmaßnahmen erfordern, werden detailliertere Annahmen über die jeweiligen Fehlfunktionen gemacht. Auf Ebene der konkreten Implementierung schließlich sind es auch Entwurfsentscheidungen, wie etwa die Länge eines CRCs oder die Anzahl möglicher Sequenznummern, welche die Menge der tolerierbaren Fehler vorgeben. Aussagen zur Fehlertoleranz werden folglich an solchen Stellen in Abhängigkeit von den entsprechenden Parametern angegeben.

Für die gesamte Arbeit ausgeschlossen werden absichtliche Angriffe als Fehlerquellen, so dass etwa die Möglichkeit zum Brechen von Signaturen nicht betrachtet wird. Sollten sich Angriffsszenarien für die hier betrachteten Systeme als relevant erweisen, so sind u. U. stärkere kryptografische Verfahren zu verwenden. Da das Konzept der Entfernten Redundanz an diesem Punkt generisch ist, steht seine Verwendung jedoch nicht im Widerspruch zu entsprechenden Anforderungen.

Durchweg nicht berücksichtigt werden ebenso Entwurfsfehler. Diesen ist stets durch spezifische Methoden (diversitärer, also mehrfacher Entwurf) zu begegnen, wobei Entwurfsfehler in keinem generellen Zusammenhang zu dem in dieser Arbeit vorgestellten Konzept der Entfernten Redundanz stehen, insbesondere also von ihr nicht per se schlechter toleriert werden als bei anderen Ansätzen.

Nicht betrachtet werden schließlich ebenfalls sog. byzantinische Fehler, durch die bei den Empfängern von Nachrichten eine unterschiedliche Sicht auf die vom Sender übermittelten Daten bzw. den Systemzustand entsteht. Je nach Redundanzgrad ist Entfernte Redundanz zwar auf die Abwesenheit solcher Fehler angewiesen, es sind jedoch Bussysteme verfügbar, welche ihr Vorkommen äußerst unwahrscheinlich machen. Sofern entsprechende Systeme nicht genutzt werden oder hinsichtlich ihrer Eigenschaften nicht ausreichen, existieren aus der Fehlertoleranzforschung formal verifizierte Übereinstimmungsprotokolle, etwa in (Lamport, Shostak, & Pease, 1982), die es erlauben, byzantinische Fehler zu tolerieren und auf denen Entfernte Redundanz aufsetzen kann.

3 Redundanz als Grundlage und Kostenfaktor der Fehlertoleranz

Es ist unmittelbar einsichtig, dass es zur Gewährleistung jeder Form von Fehlertoleranz Mittel bedarf, welche für die Erbringung der eigentlichen Nutzfunktion nicht notwendig wären, im Fehlerfall jedoch die Fehlerbehandlung realisieren oder aber das Vorliegen eines Fehlers erst erkennbar machen. Diese zusätzlichen Mittel bilden somit die Grundlage der Fehlertoleranz und werden als **Redundanz** bezeichnet:

„Redundanz bezeichnet das funktionsbereite Vorhandensein von mehr technischen Mitteln, als für die spezifizierten Nutzfunktionen des Systems benötigt werden.“ (NTG, 1982), zitiert nach (Echtle, 1990, S. 49)

Der Begriff sagt zunächst nichts über die Art der technischen Mittel und ihre Verwendung aus. Hinsichtlich beider Dimensionen gibt es also einen Entscheidungsspielraum, der nicht nur die Fehlertoleranzeigenschaften, sondern auch die Kosten des Systems berührt. Wenngleich in der Praxis eine strikte Trennung nicht möglich ist, kann es daher lohnend sein, zunächst nach dem (vornehmlich) eingesetzten redundanten Mittel zu unterscheiden. Dementsprechend unterscheidet man üblicherweise, etwa in (Echtle, 1990), nach den folgenden Merkmalen:

- **Strukturelle Redundanz** als die Verwendung von Hardwarekomponenten, welche für den reinen Nutzbetrieb nicht notwendig wären. Dies können sowohl exakte Kopien bereits vorhandener Komponenten (z. B. zwei identische Prozessoren, auf denen dieselbe Anwendung ausgeführt wird) oder zusätzliche, andersartige Komponenten (etwa eine Schaltung zum Vergleich der Ausgabe der beiden Prozessoren) sein. Da strukturelle Redundanz i. d. R. mit erheblichen Kosten verbunden ist, wird oft – so auch in dieser Arbeit – versucht, diese möglichst gering zu halten.

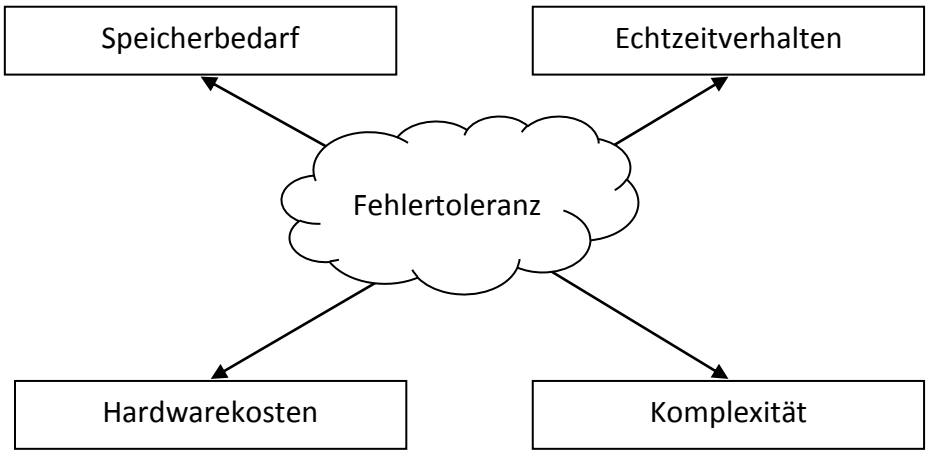
- **Funktionelle Redundanz** bezeichnet die Bereitstellung von Funktionalität, welche im Nutzbetrieb nicht benötigt wird. Als Beispiele hierfür können Diagnoseroutinen, wie z. B. der nach dem Einschalten von Rechnern durchgeführte „Power-on self-test“ (POST), oder auch die in Programmiersprachen üblichen Ausnahmebehandler dienen. Ebenfalls unter den Begriff funktionelle Redundanz werden diversitäre Exemplare von

Funktionen gefasst, d. h. deren mehrfache, verschiedenartige Implementierung zum Zwecke der Tolerierung von Entwurfsfehlern.

- **Informationsredundanz** dient der Bereitstellung von über die Nutzinformation hinausgehenden Informationen. Dies kann der Entdeckung, ggf. sogar auch der Korrektur von Fehlern dienen. Beispiel hierfür ist etwa das Anhängen einer Prüfsumme (im einfachsten Fall ein sog. „Paritätsbit“) an die Nutzinformation zur Erkennung von Übertragungsfehlern oder die (natürlich ebenso mit zusätzlicher Hardware verbundene) redundante Speicherung von Information in einem RAID-System.

- **Zeitredundanz** beschreibt das Vorhandensein zusätzlicher Zeit für Fehlertoleranzzwecke. So ermöglicht es z. B. die mehrfache Ausführung der Nutzfunktion mit anschließendem Vergleich, temporäre Fehler zu erkennen. Auch die für das Ausführen von Diagnosefunktionen benötigte Zeit fällt unter diese Form der Redundanz.

Diese hier beschriebenen Formen der Redundanz lassen sich innerhalb eines Gesamtsystems in unterschiedlicher Weise kombinieren, um in Abhängigkeit von der konkreten Anwendung eine möglichst gute Kosten-/Nutzenrelation zu erzielen und dabei die benötigte Fehlertoleranz zu gewährleisten. Insgesamt ist jedoch typischerweise ein Kompromiss zwischen den in Abbildung 1 dargestellten Zielen notwendig.



in Anlehnung an (Sneed, 1987)

Abbildung 1: „Teufelsquadrat“ der Fehlertoleranz

Ein sehr schnelles Fehlertoleranzverfahren stellt etwa die Fehlermaskierung in sog. TMR[3]- oder 2-von-3-Systemen dar: Die Nutzfunktion wird stets von drei Komponenten parallel erbracht, die Ergebnisse von einem sog. „Voter“ miteinander verglichen. Unter der Annahme, dass höchstens eine Komponente fehlerhaft ist, kann mit geringem Zeitaufwand durch Ermitteln der Mehrheitsverhältnisse ein richtiges Ergebnis ausgewählt werden (jedoch mit erheblichem Mehraufwand bzgl. der notwendigen Hardware im Vergleich zu einem nicht fehlertoleranten System). Umgekehrt würde eine mehrfache Hintereinanderausführung der Berechnung auf einer einzigen Komponente mit anschließendem Vergleich nur auf Kosten eines höheren Zeitbedarfs möglich sein. Das vollständige, mehrfach redundante Abspeichern von Informationen zu Fehlertoleranzzwecken lässt Fehler durch einen leicht zu realisierenden Vergleich vor Verwendung der entsprechenden Daten erkennen. Ist hingegen ein möglichst geringer Speicherbedarf das Optimierungskriterium, so sind komplexere Kodierungsverfahren notwendig, um weiterhin einen hohen Anteil möglicher Fehler erkennbar zu machen.

Ausgangspunkt der vorliegenden Arbeit im oben dargestellten Spannungsfeld sind die durch strukturelle Redundanz verursachten Hardwarekosten. Abhängig davon, ob Sicherheit oder Zuverlässigkeit (vgl. Kapitel 2) das angestrebte Entwurfsziel ist, lässt sich dieser Aufwand zahlenmäßig in allgemeiner Form wie folgt fassen:

- Wenn eine möglichst hohe Sicherheit erzielt werden soll, beinhaltet dies i. d. R. nicht die Forderung, dass der Nutzbetrieb des Systems im Fehlerfall fortzusetzen ist. Es reicht also aus, Fehler lediglich zu erkennen, um anschließend das System in einen sicheren Zustand zu überführen (es also z. B. abzuschalten). Eine solche Fehlererkennung durch Vergleich ist als Einstimmigkeitsentscheidung (Lala, 1985) bereits möglich, wenn zusätzlich zu k als fehlerhaft angenommenen Exemplaren ein weiteres fehlerfreies als Referenz zur Verfügung steht. Als Redundanz werden folglich insgesamt $m = k + 1$ Exemplare benötigt. Das System erbringt seine Funktion, solange die Zahl der fehlerfreien Exemplare n gleich der Gesamtzahl der Exemplare m ist (Echtle, 1990).
- Sobald Zuverlässigkeit gefordert wird, in dem Sinne, dass die Aufrechterhaltung des Nutzbetriebs auch im Fehlerfall zu gewährleisten ist, ist eine Fehlererkennung nicht ausreichend. Es muss in diesem Fall k fehlerhaften Exemplaren eine absolute Mehrheit von fehlerfreien Exemplaren gegenübergestellt werden, um die fehlerhaf-

[3] Triple Modular Redundancy

ten Exemplare „überstimmen“ zu können. Aus Aufwandsgründen wird man hier die kleinstmögliche Anzahl $n = k + 1$ wählen, so dass als Redundanz insgesamt $m = 2 \cdot k + 1$ Exemplare benötigt werden. Ein solches System wird als **n-von-m-System** bezeichnet, wobei n fehlerfreie Exemplare dazu dienen, Fehler in $m - n$ fehlerhaften Exemplaren zu tolerieren (Siewiorek & Swarz, 1982).

Wenn der so entstehende Mehraufwand als zu hoch bewertet wird, ist es ein möglicher Ansatz, auf ein anderes der oben beschriebenen *redundanten Mittel* auszuweichen. So kann als Ersatz für strukturelle Redundanz zusätzliche Zeit dienen, wenn etwa misslungene Operationen nach Fehlererkennung erneut durchgeführt werden. Informationsredundanz ermöglicht das Wiederaufsetzen auf einem (hoffentlich) fehlerfreien Systemzustand, während die Implementierung dieser Verfahren funktionelle Redundanz darstellt.

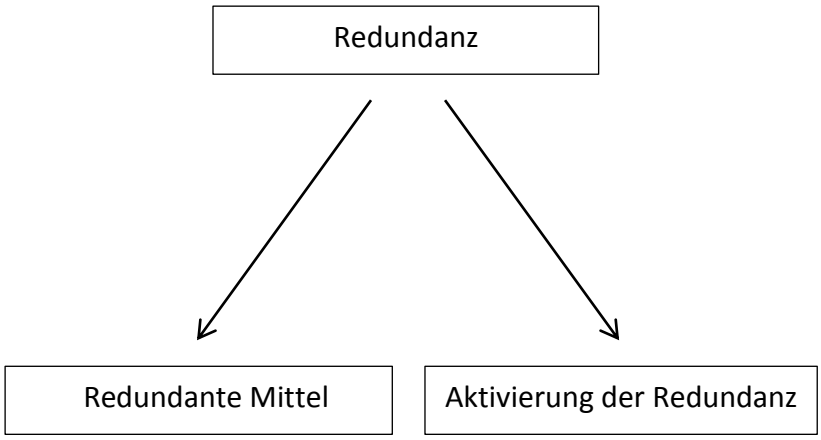
Ein anderer Ansatz besteht darin, den *Zeitpunkt der Verwendung* redundanter Mittel zu variieren, um ihre bessere Auslastung zu erreichen. Bezogen auf die Aktivierung redundanter Mittel unterscheidet (Echtle, 1990) demgemäß unabhängig von der Art der eingesetzten Mittel:

- **Statische Redundanz** als Bezeichnung für redundante Mittel, die permanent zu Fehlertoleranzzwecken vorhanden sind und in dieser Funktion genutzt werden, auch wenn noch kein Fehler eingetreten ist.

- **Dynamische Redundanz** in Form einer Aktivierung der redundanten Mittel zu Fehlertoleranzzwecken erst nach Auftreten eines Fehlers. Eine genauere Unterscheidung liefert:
  - **ungenutzte Redundanz:** Die redundanten Mittel bleiben bis zum Fehlereintritt inaktiv.
  - **fremdgenutzte Redundanz:** Nutzung der Mittel bis zum Fehlereintritt für nicht fehlertoleranzbezogene Funktionen.
  - **gegenseitige Redundanz:** Redundante Komponenten dienen wechselseitig als Ersatz füreinander.

- **Hybridredundanz** bei Kombination beider Konzepte, d. h. einer Neuzuordnung statisch redundanter Mittel im Fehlerfall.

Mit den bislang dargestellten Möglichkeiten ergibt sich insgesamt die in Abbildung 2 dargestellte Situation als Spielraum für die Verwendung von Redundanztechniken beim Entwurf fehlertoleranter Systeme:



modifiziert übernommen aus (Echtle, 1990)

Abbildung 2: Klassifizierung von Redundanztechniken

Wenn über das hierdurch gegebene Maß hinaus Wege zu einer Reduzierung des Aufwands für strukturelle Redundanz eröffnet werden sollen, stellt sich somit die Frage nach weiteren Freiheitsgraden.

Das im Folgenden vorgestellte Konzept der Entfernten Redundanz nutzt hierfür den *Ort des Vorhandenseins* redundanter Mittel. Das damit erreichbare Einsparpotenzial hängt dann von weiteren Faktoren ab, u. a. von der Existenz freier Rechenkapazitäten im Gesamtsystem. Im günstigsten Fall jedoch ist eine Reduzierung auf 1 statt m zusätzlicher Knoten bei fehlererkennenden bzw. n statt m Knoten bei fehlertolerierenden Systemen möglich. Unabhängig davon eröffnen sich in jedem Fall neue Gestaltungsmöglichkeiten für den Systementwurf, welche in ausführlicher Form Gegenstand des nun nachfolgenden Kapitels sein werden.

4 Das Konzept der Entfernten Redundanz

4.1 Grundidee und Entwurfsprinzipien

Die Grundidee Entfernter Redundanz liegt in der Entkopplung vom Ort des Vorhandenseins redundanter Komponenten und dem Ort der Nutzung dieser Redundanz in Bezug auf fehlertolerante Steuerungssysteme, welche aus einer Vielzahl von Sensoren, Aktuatoren und Rechnern bestehen. Diese Idee geht in ihren Grundzügen zurück auf Arbeiten des Autors im EU-Projekt EASIS[4] (EASIS, 2006) und wurde in der Folge am Lehrstuhl „Verlässlichkeit von Rechensystemen“ an der Universität Duisburg-Essen weiter ausgearbeitet, vgl. etwa (Echtle, Kimmeskamp, Jacquet, Malassé, Pock, & Walter, 2010).

Ziel des Konzepts ist es, die in fehlertoleranten Steuerungen typischerweise bestehenden mehrfachen direkten Kabelverbindungen zwischen Steuergeräten und Peripherie so weit wie möglich aufzulösen und durch einen indirekten Zugriff über fremde Komponenten hinweg zu ersetzen. Für einen solchen Zugriff sind jedoch Schutzmaßnahmen in Form digitaler Signaturen erforderlich, damit Verfälschungen von Informationen durch weiterleitende fremde Komponenten für den Empfänger erkennbar sind. Gelingt dies, so können auch sicherheitskritische Anwendungsfunktionen prinzipiell auf beliebigen Steuergeräten im Netzwerk ausgeführt werden, selbst wenn sie auf Peripherie (also Sensoren und Aktuatoren) zugreifen müssen, die nicht direkt an das „eigene“ Steuergerät angeschlossen ist. Sofern letztlich die entsprechenden Rechenkapazitäten vorhanden sind, können Steuergeräte hierdurch prinzipiell mehrere Anwendungen beherbergen, sogar, wenn diese unterschiedliche Regelkreise betreffen. Die Zahl der Steuergeräte kann so auf das zur fehlertoleranten Ansteuerung der Peripherie unbedingt notwendige Maß reduziert werden. Die nachfolgende Abbildung 3 fasst die sich aus den oben genannten Punkten ergebenden Entwurfsprinzipien Entfernter Redundanz zunächst grafisch zusammen.

[4] Electronic Architecture and System Engineering for Integrated Safety Systems

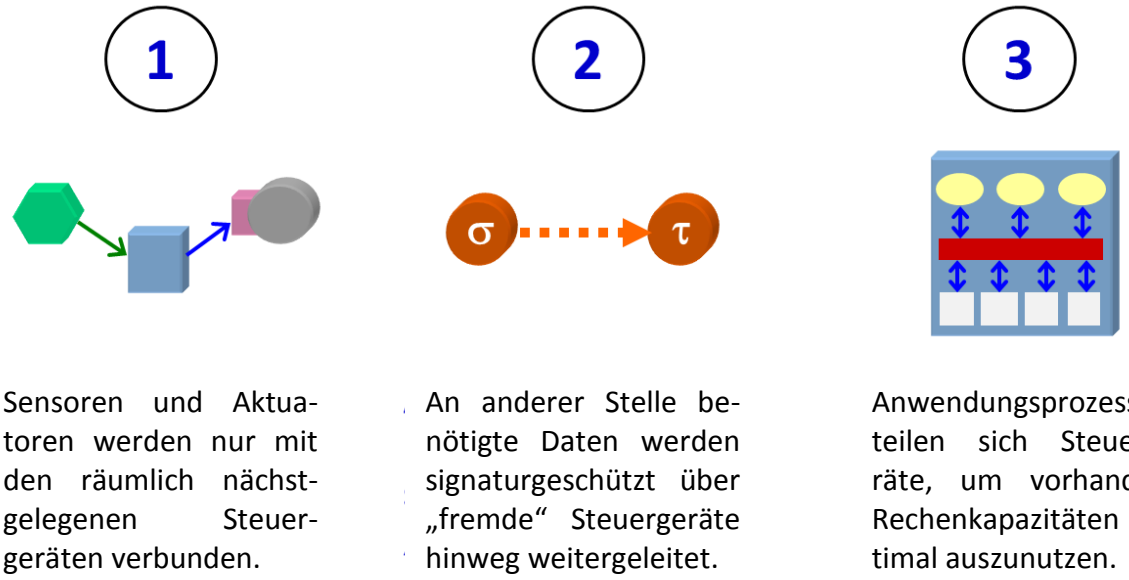


Abbildung 3: Entwurfsprinzipien Entfernter Redundanz

Da die hier abgebildeten Grundgedanken, die man durchaus auch als Schritte in einem Entwurfsprozess verstehen könnte, für Entfernte Redundanz von wesentlicher Bedeutung sind, sollen die folgenden Erläuterungen die Hintergründe nochmals detaillierter beschreiben:

1. Üblicherweise werden in fehlertoleranten Systemen Sensoren und Aktuatoren mit mehreren Steuergeräten verbunden. Die Ursache hierfür liegt in der notwendigerweise redundanten Verarbeitung der entsprechenden Sensordaten, bzw. in der redundanten Ansteuerung der Aktuatoren. Diese Notwendigkeit besteht auch bei Entfernter Redundanz. Hier werden jedoch Sensoren und Aktuatoren nur an das räumlich nächstgelegene Steuergerät in einem Netzwerk angeschlossen, selbst wenn die von diesen Komponenten ausgehenden oder für sie bestimmten Daten letztendlich (auch) in einem oder mehreren anderen Steuergerät(en) verarbeitet werden. Es entfällt damit die für fehlertolerante Systeme ansonsten typische Verkabelung „über Kreuz“.
2. Sofern Ein-/Ausgabedaten an anderer Stelle benötigt werden, werden sie digital signiert über fremde Steuergeräte hinweg weitergeleitet. Da in den entsprechenden Systemen die meisten Steuergeräte üblicherweise ohnehin über Bussysteme miteinander vernetzt sind, existiert hierfür bereits ein Großteil der hardwareseitigen Voraussetzungen. Durch die digitale Signatur ist gewährleistet, dass Informationen auf diesem Weg mit sehr hoher Wahrscheinlichkeit nicht unerkannt verfälscht werden können. Zur Erzeugung und Prüfung der Signatur sind zu-

sätzliche Komponenten notwendig, auf welche im weiteren Verlauf dieser Arbeit eingegangen werden wird. Konzeptionell ist hiermit die Voraussetzung dafür geschaffen, dass eine fehlertolerante Ansteuerung „entfernt redundanter“ Komponenten möglich ist.

3. Durch den Wegfall der direkten Verkabelung mit Peripheriekomponenten wird sich oftmals zeigen, dass bestimmte Steuergeräte nunmehr ausschließlich noch mit dem Bussystem verbunden sein müssen. Das Anwendungsprogramm eines solchen Steuergeräts kann folglich prinzipiell auch auf anderen, ggf. sogar bereits existierenden Knoten im Netzwerk ausgeführt werden. Wo immer dies möglich ist, bietet es sich also an, vorhandene Rechenkapazität in Steuergeräten auch über verschiedene Regelkreise des Gesamtsystems hinweg gemeinsam zu nutzen. Da der Ort der Funktionserbringung durch Entfernte Redundanz variabel wird, können so Steuergeräte, aber auch Peripheriekomponenten eingespart werden, ohne dass die Fehlertoleranzeigenschaften des Systems beeinträchtigt werden.

Der sich durch Anwendung des Konzepts der Entfernten Redundanz im System letztlich ergebende logische Ablauf betrifft typischerweise zwei Situationen:

- Von einem Sensor erfasste Daten sind über einen fremden Rechner an den eigentlichen Zielrechner, welcher diese Daten verarbeitet, weiterzuleiten.
- Steuerbefehle eines Rechners sind über einen fremden Rechner an den anzusteuernden Aktuator weiterzuleiten.

Beide Abläufe erfordern den Einsatz von digitalen Signaturen, um Verfälschungen durch den jeweils fremden Rechner zu verhindern und sind in der nachfolgenden Abbildung 4 dargestellt.

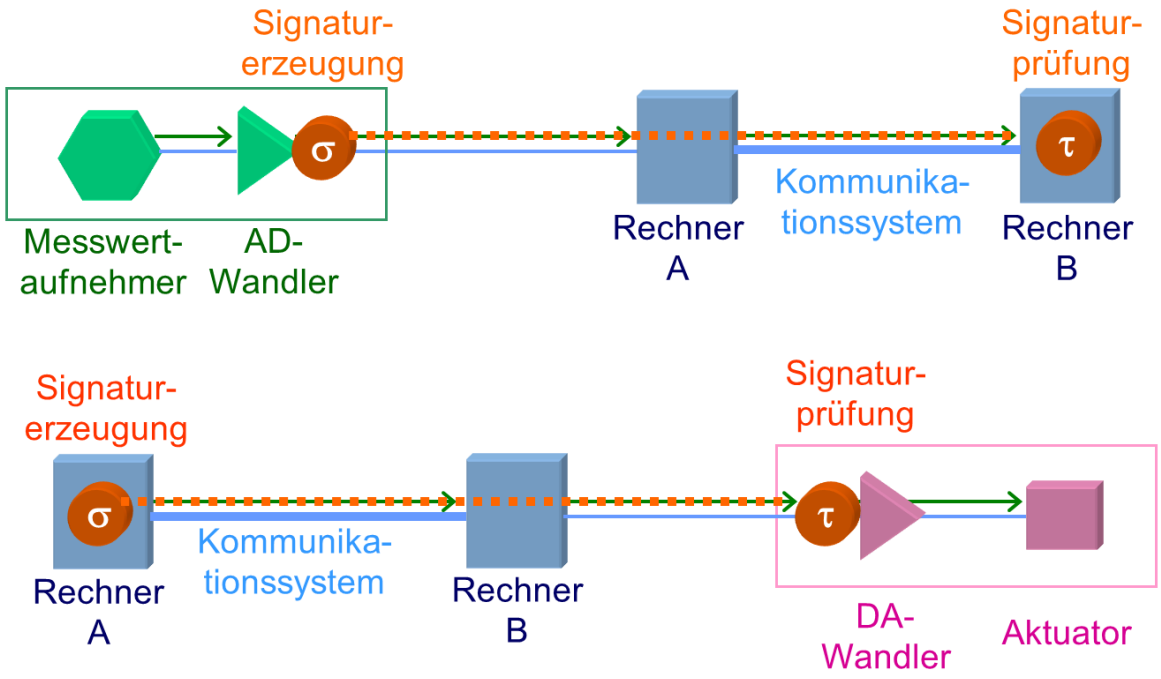


Abbildung 4: Abläufe bei der Verwendung digitaler Signaturen

In dem im oberen Teil der Abbildung dargestellten Prozess werden die von einem Sensor erfassten Daten zunächst digitalisiert und mit einer Signatur versehen (Funktion σ). Die so geschützten Daten werden sodann an den mit dem Sensor verbundenen Rechner A übermittelt. Rechner A leitet die Daten im fehlerfreien Fall unverändert weiter über ein Kommunikationssystem an den eigentlichen Zielrechner B. Dieser prüft, ob die Signatur bei der Weiterleitung beschädigt wurde (Funktion τ), bevor die Daten nach bestandenem Test verarbeitet werden.

Der umgekehrte Weg ist im unteren Teil der Abbildung dargestellt: Steuerbefehle von Rechner A sind für einen Aktuator bestimmt. Noch vor der Versendung werden sie digital signiert und dann über ein Kommunikationssystem an Rechner B gesendet. Sofern nicht selbst defekt, leitet dieser Rechner die Daten weiter an den mit ihm verbundenen Aktuator. Von einer dem Aktuator vorgeschalteten Elektronik wird zunächst die Signatur geprüft, bevor der Steuerbefehl nach bestandenem Test dem Aktuator zur Verfügung gestellt wird.

In beiden Fällen kann die Kommunikation auch über mehrere weiterleitende Rechner geschehen, etwa wenn Rechner A und Rechner B sich in unterschiedlichen Netzwerken befinden, welche über ein Gateway miteinander verbunden sind. Diese Situation wird in der vorliegenden Arbeit nicht behandelt, da sie keinen prinzipiellen Einfluss auf das Verfahren hat.

Wenn ein Signaturtest nicht bestanden wird, muss pessimistisch vorgegangen werden: Ein Sensorwert wird als fehlerhaft, ein Wert zur Ansteuerung eines Aktuators als Passivierungsaufforderung interpretiert. Damit einzelne Fehler nicht unzulässig viele Passivierungen hervorrufen können, müssen die möglichen Mehrheitsverhältnisse von fehlerfreien und fehlerhaften Komponenten genau geplant und durch entsprechende Votingverfahren berücksichtigt werden (beschrieben für verschiedene Systemklassen in Kapitel 4.3).

4.2 Signaturverfahren und gemeinsame Nutzung von Steuergeräten

Bei der Durchleitung von Daten durch fremde Knoten kann nicht verhindert werden, dass Daten von einem fehlerhaften Knoten verfälscht werden. Ebenso kann nicht verhindert werden, dass durch einen solchen Knoten spontan eigene Daten oder überhaupt keine Daten gesendet werden.

Es ist jedoch möglich, Maßnahmen zur Erkennung dieser Fehler zu treffen, um anschließend in geeigneter Form darauf reagieren zu können. Welche Verfahren im konkreten Fall hierfür verwendet werden, hängt typischerweise nicht nur von der Vorgabe der zu tolerierenden Fehler, sondern auch von den durch die Anwendung selbst vorgegebenen Rahmenbedingungen ab. Die dabei relevanten Kriterien können etwa umfassen:

- leichte Implementierbarkeit,
- kurze Ausführungsdauer,
- geringer Speicherbedarf,
- Konformität mit Standards.

Einen guten Schutz gegenüber zufälligen (d. h. fehlerinduzierten) Verfälschungen insbesondere durch Rauschen des Übertragungskanals bieten CRCs[5] (Peterson & Brown, 1961). Abhängig vom Anwendungsgebiet existieren oftmals von Gremien empfohlene bzw. in Standards aufgenommene Empfehlungen für die Wahl geeigneter Generatorpolynome. Zusätzliche Anforderungen, wie die Gewährleistung von Integrität und Authentizität von Nachrichten auch bei bewussten Manipulationen, erfordern jedoch in jedem Fall darüber hinausgehende kryptographische Verfahren. Das Konzept der Entfernten Redundanz ist in dieser Hinsicht generisch und schreibt kein bestimmtes Verfahren vor. Vor dem Hintergrund ent-

[5] Cyclic Redundancy Check

sprechender Bestrebungen im Automobilbereich wird allerdings mit dem im Folgenden beschriebenen Verfahren eine Methode vorgestellt, welche bei geringem Aufwand einen relativ umfassenden Schutz bietet.

Das hier vorgestellte, auf einem Public-Key-Verfahren basierende Schema (Echtle, 2003), (Echtle & Kimmeskamp, 2009) besteht aus drei Schritten: Über die zu schützenden Daten wird zunächst ein herkömmlicher CRC gebildet. Dieser wird sodann mit einem geheimen Schlüssel signiert. Den Empfängern werden öffentliche Schlüssel bereitgestellt, mit denen sie die Signatur überprüfen können. Um den Spezialfall eines fehlerbedingten wiederholten Sendens alter, aber gültig signierter Nachrichten zu behandeln, kommen zusätzlich fortlaufende Sequenznummern zum Einsatz.

Alle Berechnungen finden modulo m statt, wobei m typischerweise eine Zweierpotenz ist. Zunächst werden zwei Faktoren a und b zufällig aus dem Intervall $[0, m-1]$ gewählt und das Produkt $c \coloneqq a \cdot b \ (mod\ m)$ berechnet[6]. Der Faktor a verbleibt für die Signaturerzeugung beim Sender, während b und c zur Überprüfung der Signatur allen potentiellen Empfängern bereitgestellt werden. Die Schlüsselverteilung kann dabei statisch bei der Erzeugung des Systems erfolgen. Vom Wert m hängen sowohl Aufwand als auch potentielle Sicherheit des Verfahrens ab: zu kleine Werte könnten die Anzahl möglicher unterschiedlicher Schlüssel zu gering werden lassen, zu große Werte führen zwangsläufig zu einem höheren Übertragungsaufwand. Auch die für a und b gewählten Zahlen haben einen Einfluss auf die Güte des Verfahrens; dies wird Gegenstand eines späteren Kapitels sein.

Seien nun d die zu übertragenden Nutzdaten und n die fortlaufende Sequenznummer. Die Signaturfunktion σ wird definiert als:

$$\sigma(n,d) \coloneqq CRC\big(concat(n,d)\big) \cdot a \quad (mod\ m)$$

Die Operation $concat$ steht hierbei für die Konkatenation von Bitvektoren. Der durch die Funktion τ gegebene zugehörige Signaturtest wird definiert als:

$$\tau(n,d,s) \coloneqq \big(s \cdot b = CRC\big(concat(n,d)\big) \cdot c\big) \quad (mod\ m)$$

[6] Da die Wirksamkeit des Verfahrens darauf basiert, dass der Umkehrschluss a = c/b (mod m) nicht gilt, ist es empfehlenswert, zu kleine Werte mit a · b < m auszuschließen.

Hierbei steht s für die empfangene Signatur. Der Korrektheitsbeweis ergibt sich unmittelbar aus dem Einsetzen von $s = CRC\big(concat(n,d)\big) \cdot a$ und $c = a \cdot b$ in vorstehende Gleichung, woraus die Identität $CRC\big(concat(n,d)\big) \cdot a \cdot b = CRC\big(concat(n,d)\big) \cdot a \cdot b$ folgt.

Beispiel

- Verwendet wird $m = 1024$; die Faktoren $a = 976$ und $b = 631$ seien zufällig gewählt.
- Folglich ergibt sich $c = 976 \cdot 631\ (mod\ 1024) = 432$.
- Das zu sendende Datenwort d sei nun 451, eine Sequenznummer wird nicht verwendet.
- Die Signaturerzeugung liefert: $\sigma(451) = 451 \cdot 976(mod\ 1024) = 880$.
- Übertragen wird eine Nachricht bestehend aus Nutzdaten und Signatur $[451, 880]$.
- Die Überprüfung beim Empfänger liefert: $\tau = (880 \cdot 631(mod\ 1024) = 451 \cdot 432(mod 1024))$, was $272 = 272$, also „$true$“ ergibt, die Signatur ist somit als gültig erkannt.

Signaturen bieten einen Schutz gegenüber Verfälschungen bei der Durchleitung von Daten durch fremde Steuergeräte. Darauf aufbauend ermöglicht es Entfernte Redundanz, Anwendungen unterschiedlicher Regelkreise gemeinsam auf einem Steuergerät zu betreiben, da es nun nicht mehr notwendig ist, Sensoren und Aktuatoren unmittelbar mit demjenigen Steuergerät zu verbinden, auf dem die zugehörige Anwendung abläuft. Bei genügend Rechenkapazität reichen somit prinzipiell drei Steuergeräte aus, um beliebig viele Regelkreise fehlertolerant anzusteuern, ohne (!) dass weitere Schutzmaßnahmen notwendig wären. So lange sich Fehler nur auf ein Steuergerät auswirken, bleibt eine fehlerfreie Mehrheit von zwei Steuergeräten bestehen, unabhängig davon, wie viele Anwendungen auf ihnen ausgeführt werden. Nicht toleriert werden jedoch Entwurfsfehler, durch welche alle drei Steuergeräte zugleich außer Funktion gesetzt würden.

Wenn das Betriebssystem entsprechende Schutzmechanismen bietet, können allerdings Entwurfsfehler in der Anwendungssoftware eines Regelkreises auf diesen beschränkt

bleiben, ohne dass die anderen Regelkreise hierdurch beeinträchtigt würden. Ein solcher Schutz müsste die im Folgenden aufgeführten Aspekte berücksichtigen:

- **Speicher:** keine Anwendung darf, z. B. in Folge von Programmierfehlern, schreibend auf den Speicher einer anderen Anwendung zugreifen. Eine übliche Maßnahme, um dies zu erreichen, ist die Verwendung einer sog. Memory Management Unit (MMU).
- **Laufzeit:** durch das Scheduling-Verfahren des Betriebssystems ist sicherzustellen, dass alle Anwendungen genügend Laufzeit erhalten. Dies kann durch eine statische Vorabbestimmung der maximalen Bedarfe (WCET[7]-Analyse) und eine Überwachung der tatsächlichen Laufzeit („Deadline-Monitoring“) verbunden mit einem geeigneten Scheduling-Verfahren (z. B. Deadline-Scheduling) während des Betriebs erfolgen.
- **Zugriff auf Ressourcen:** Gemeinsam genutzte weitere Ressourcen (Kommunikationskanal, Sensoren, Aktuatoren) dürfen durch eine fehlerhafte Anwendung nicht blockiert werden. Auch dies erfordert präventive Maßnahmen zur Verklemmungsverhinderung (am einfachsten durch Reservierung von Ressourcen, z. B. von Zeit-Slots im Kommunikationssystem FlexRay).

Entwurfsfehler des Betriebssystems können durch keine der genannten Maßnahmen toleriert werden. Hierfür wäre vielmehr ein diversitärer Entwurf des Betriebssystems notwendig. Da dies jedoch generell und nicht nur im Besonderen für entfernte Redundanz gilt, wird dieser Aspekt in der vorliegenden Arbeit nicht behandelt.

Ebenfalls ausgeklammert bleibt die Situation mehrerer miteinander interagierender Anwendungen. Für eine Beschreibung der in einem solchen Fall notwendigen Schutzmaßnahmen sei auf (EASIS, 2006) verwiesen. Aus Gründen der Übersichtlichkeit und zum Zwecke der Fokussierung auf den Kern des Konzepts gehen die folgenden Kapitel stattdessen stets von Systemen mit nur einem einzigen Regelkreis aus und beschreiben anhand derer die Realisierung fehlertoleranter Systeme mit Entfernter Redundanz für verschiedene Fehlertoleranzanforderungen.

[7] Worst case execution time

4.3 Architektur eines fehlertoleranten Gesamtsystems

4.3.1 Nichtredundantes System

Die in den folgenden Kapiteln angestellten Überlegungen zum Aufbau eines fehlertoleranten Gesamtsystems mit Entfernter Redundanz nehmen ihren Ursprung in einer nichtredundanten (und daher auch nicht fehlertoleranten) Steuerung. Ein typisches Beispiel für ein solches System ist in Abbildung 5 dargestellt:

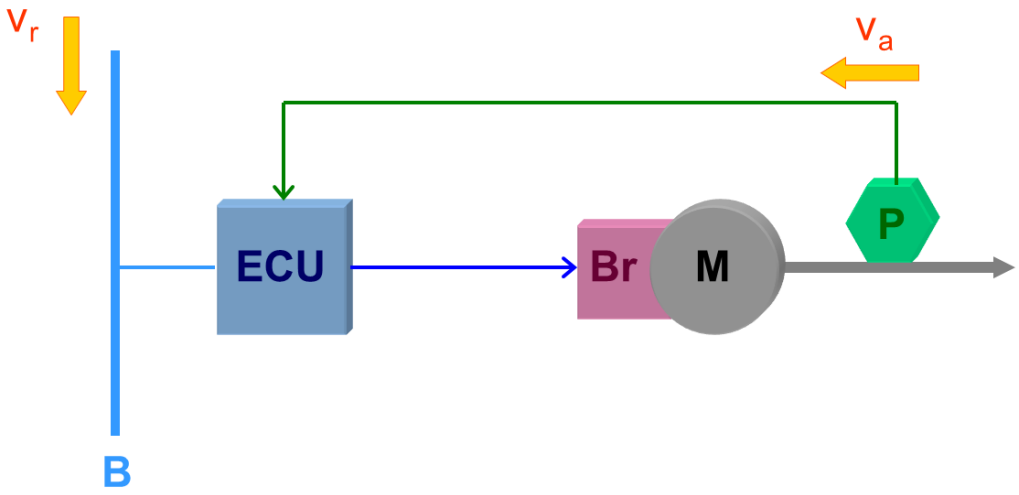


Abbildung 5: Nichtredundante Ansteuerung eines Motors

Ein Sollwert v_r (reference value) wird über einen Datenbus B an ein Steuergerät (ECU, electronic control unit) übertragen. Dieses wirkt über eine Brückenschaltung Br zur elektrischen Ansteuerung auf den angeschlossenen Motor M ein. Durch den Motor wird eine hier durch einen grauen Pfeil angedeutete, ansonsten jedoch nicht näher betrachtete Mechanik bewegt. Die Position der so angesteuerten Mechanik wird als Istwert v_a (actual value) von einem Positionssensor P ermittelt und zurück an die ECU übertragen, wo sich durch Berechnung der Regeldifferenz $v_r - v_a$ zur Ermittlung des nun nachfolgenden Steuerbefehls der Regelkreis schließt.

Je nach Art der Anwendung könnten auch andere Arten von Aktuatoren, wie etwa Ventile oder Hydraulikzylinder sowie andere Sensoren zum Einsatz kommen. Das Grundprinzip des Systemaufbaus ist jedoch hiervon unberührt und wird in dieser Form auch in den folgenden (fehlertoleranten und somit Redundanz beinhaltenden) Systemvarianten beibehalten.

Aus Zuverlässigkeitssicht ist an dieser Stelle festzuhalten, dass das System in der abgebildeten Form zunächst nicht fehlertolerant ist. Vielmehr führt jeder einzelne Fehler in

einer beliebigen Komponente zum Systemausfall. Abgesehen von dem Bestreben, die einzelnen Komponenten möglichst zuverlässig zu konstruieren, bleibt folglich nur die Möglichkeit, redundante Komponenten zu verwenden, wenn die Zuverlässigkeit des Gesamtsystems erhöht werden soll oder muss.

4.3.2 Fail-safe-Konfiguration

Eine erste Stufe zur Fehlertoleranz stellen so genannte Fail-safe-Systeme (Gschwind & Uebel, 1984) dar. Diese Klasse von Systemen garantiert eine Erbringung der Nutzfunktion lediglich bei Abwesenheit von Fehlern. Beim Auftreten von Fehlern wird jedoch ein Ausfallverhalten verlangt, welches das System in einen sicheren Zustand, also z. B. einen Stillstand, überführt bzw. es in einem solchen Zustand belässt.

In Bezug auf die in Kapitel 2 vorgestellten Eigenschaften fehlertoleranter Systeme wird somit von Fail-safe-Systemen lediglich Sicherheit verlangt. Die Forderung nach Zuverlässigkeit wird bewusst nicht gestellt. Fail-Safe-Systeme können allerdings als Teil größerer Systeme eingesetzt werden, da ihr definiertes Ausfallverhalten auf höherer Ebene typischerweise günstiger behandelt werden kann als ein beliebiges Fehlverhalten. Auf diese Weise kann, bezogen auf das Gesamtsystem, der Einsatz von Fail-Safe-Systemen auch der Zuverlässigkeit dienen.

Zur Erzielung eines Fail-safe-Verhaltens lassen sich im günstigsten Fall die Gesetze der Physik nutzen: mechanische Signale im Eisenbahnverkehr fallen etwa durch die Schwerkraft automatisch in den Zustand „Halt“, sobald der entsprechende Seilzug reißt; das Reißen einer Kupplung zwischen zwei Zugteilen führt, bedingt durch den eintretenden Druckverlust, ohne weiteres Zutun zu einer Bremsung beider Zugteile. Im Allgemeinen ist jedoch für die Erreichung von Sicherheit Redundanz erforderlich.

Bevor nun gezeigt wird, wie die in Kapitel 4.3.1 vorgestellte Steuerung mit dem Konzept der Entfernten Redundanz zu einem Fail-Safe-System erweitert werden kann, soll zum Vergleich zunächst der bisherige Stand der Technik, hier als Dedizierte Redundanz bezeichnet, dargestellt werden (s. Abbildung 6). Alle hier gezeigten Systemvarianten wurden dabei auf Grundlage einer Einfehlerannahme erstellt, erlauben es also, einen einzelnen Fehler zu behandeln, wobei der Fehlerort beliebig ist.

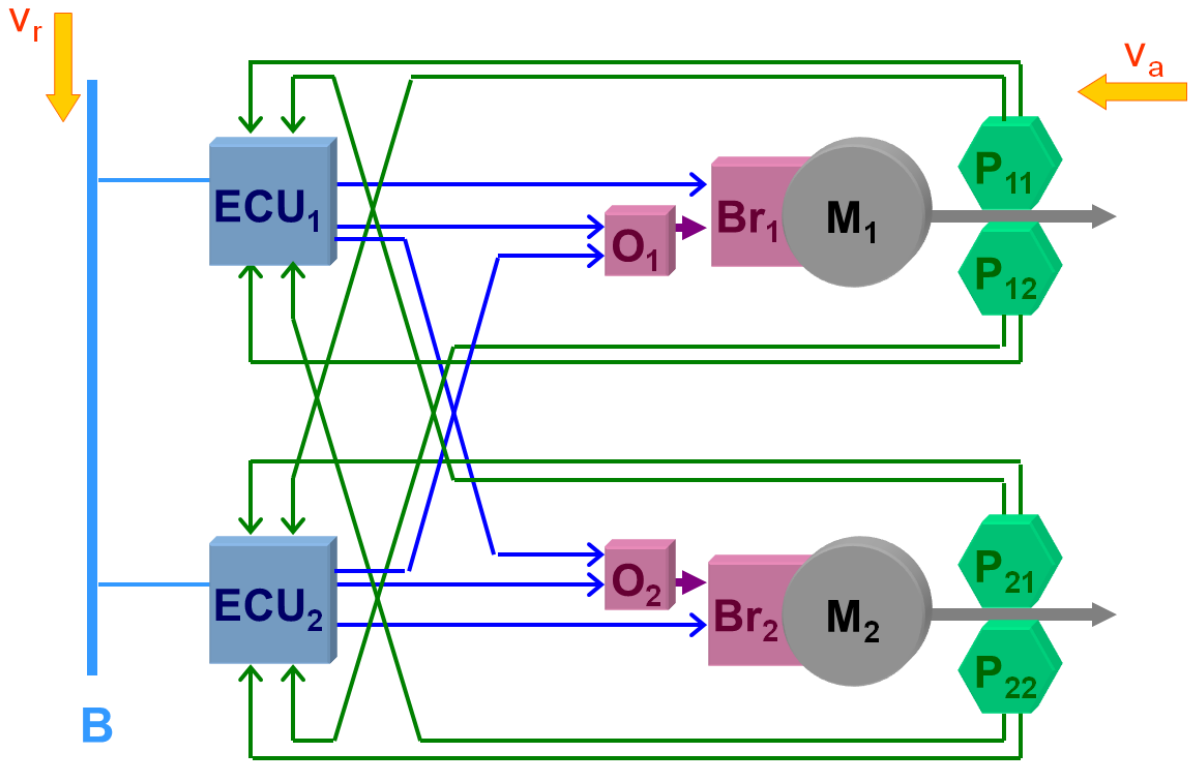


Abbildung 6: Fail-safe-Konfiguration, Dedizierte Redundanz

Das in obenstehender Abbildung als ECU_1 bezeichnete Steuergerät wirkt über die Brücke Br_1 auf Motor M_1 ein, welcher mit einer nicht näher betrachteten Mechanik (symbolisiert durch den vom Motor ausgehenden grauen Pfeil) gekoppelt ist. Der hierfür benötigte Sollwert v_r wird über ein einkanaliges Bussystem B übermittelt. Gleichermaßen steuert ECU_2 über die Brücke Br_2 den ihr zugeordneten Motor M_2 einschließlich zugehöriger Mechanik an.

Im Sinne eines Fail-safe-Verhaltens wird nun gefordert, dass die Mechanik beider Systemteile auch im Fehlerfall in annähernd derselben Position verbleibt. Um dies zu gewährleisten, ermitteln zwei Positionssensoren P_{11} und P_{12} die Stellung des oberen Teils der Mechanik und liefern den entsprechenden Istwert v_a. Für den unteren Teil des Systems wird diese Funktion analog durch P_{21} und P_{22} erbracht.

Entscheidend ist, dass aus Abweichungen zwischen den so ermittelten Werten Fehler bei der Ansteuerung erkannt werden. Bei einer Abweichung zwischen den beiden oberen Sensoren P_{11} und P_{12} ist zunächst nicht klar, welcher von den zwei unterschiedlichen Werten falsch ist. Aufgrund der Einfehlerannahme kann aber gefolgert werden, dass höchstens einer der Positionssensoren fehlerhaft und alle anderen Komponenten fehlerfrei sind. Folglich sind beide Motoren und auch die sie ansteuernden ECUs fehlerfrei. Beide Motoren haben also

tatsächlich die korrekte Position erreicht. In einem solchen Fall sind folglich die Werte der beiden unteren Sensoren hinzuzuziehen (und umgekehrt). Mit der so geschaffenen Redundanz lässt sich stets erkennen, welcher der vier Positionssensoren fehlerhaft und daher in der Folge zu ignorieren ist.

Anders verhält sich die Situation, wenn P_{11} und P_{12} übereinstimmend einen beliebigen Wert a, P_{21} und P_{22} jedoch übereinstimmend einen davon verschiedenen Wert b liefern. Durch die Einfehlerannahme ist ausgeschlossen, dass zwei Sensoren zugleich fehlerhaft sind. Folglich muss einer der beiden Motoren defekt sein oder falsch angesteuert werden. Um die Fail-safe-Eigenschaft einer (nahezu) identischen Position beider Systemteile zu erreichen, sind dementsprechend beide Motoren zu passivieren. Die Entscheidung für eine Passivierung wird von beiden ECUs getroffen und über die den Motoren vorgelagerten Endstufen (output stages) O_1 und O_2 realisiert. Jeder der beiden Motoren wird von der entsprechenden Endstufe nur dann mit Energie versorgt, wenn beide ECUs dies veranlassen. Auf diese Weise ist gewährleistet, dass auch ein sog. byzantinisches Verhalten der ECUs toleriert wird: Wenn etwa ECU_1 stets M_1 aktiviert, M_2 jedoch passiviert, so ist dies für ECU_2 nach kurzer Zeit anhand abweichender Positionswerte erkennbar und führt in der Folge zu einer Passivierung von M_1 und M_2 durch die (nach Annahme fehlerfreie) ECU_2, womit der geforderte Systemstillstand erreicht ist.

Der Preis der so erreichten Fail-safe-Eigenschaft des Systems liegt bei der im vorangegangenen Abschnitt geschilderten Verwendung Dedizierter Redundanz nicht nur in einem Mehraufwand an Hardware für redundante Komponenten. Ein wesentlicher Faktor ist über dies hinaus die für die wechselseitige Kontrolle beider Systemteile notwendige umfangreiche Verkabelung „über Kreuz“, welche die Komplexität des Gesamtsystems deutlich erhöht und eine strukturelle Abhängigkeit der beiden Systemteile voneinander erzeugt.

Die Zielsetzung Entfernter Redundanz ist es nun, diese Nachteile eines Systemaufbaus mit Dedizierter Redundanz zu vermeiden, zugleich jedoch die Fehlertoleranzeigenschaften zu erhalten. Grundlage sind somit dieselben Fehlerannahmen, Ziel ist es allerdings, das Geforderte auf eine günstigere Art und Weise zu erreichen. Wie dies im Beispiel des Fail-safe-Systems gelingen kann, wird in nachfolgender Abbildung 7 dargestellt.

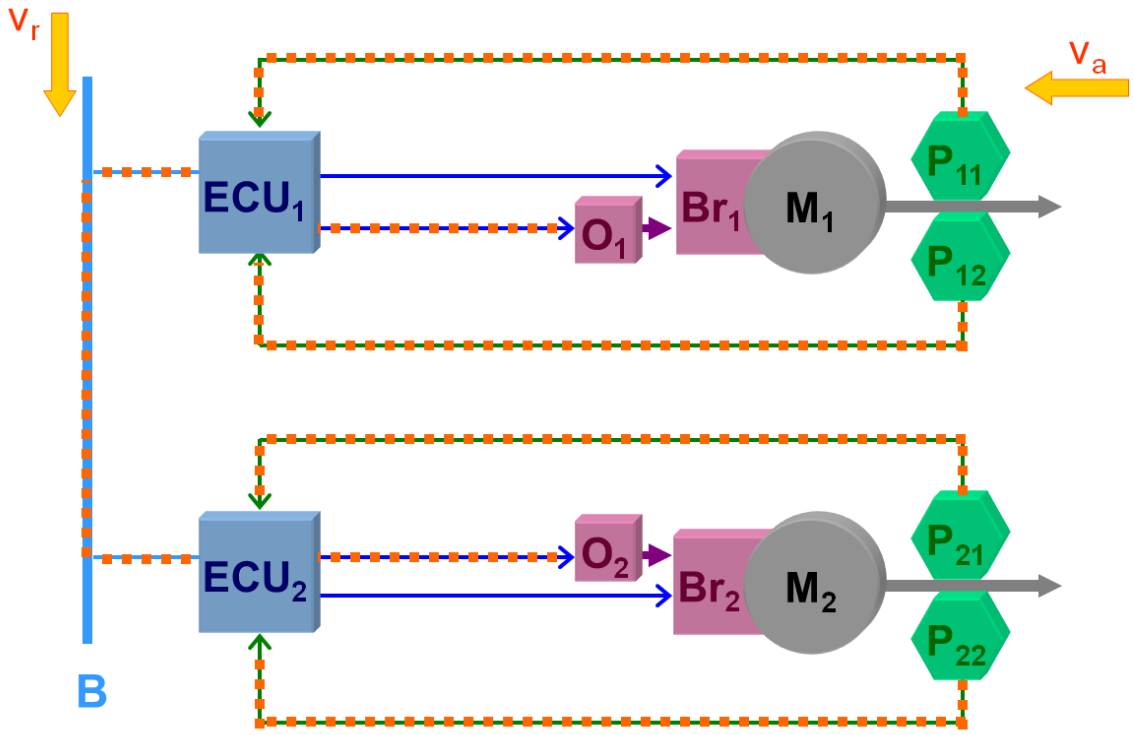


Abbildung 7: Fail-safe-Konfiguration, Entfernte Redundanz

Das oben abgebildete System mit Entfernter Redundanz enthält dieselben Komponenten, wie sie schon von der Variante mit Dedizierter Redundanz bekannt sind. Im Aufbau unterscheidet es sich jedoch dahingehend, dass die Überkreuzverkabelung zwischen Positionssensoren und ECUs bzw. zwischen ECUs und Endstufen entfällt. Sie wird ersetzt durch signaturgeschützte Kommunikation, welche in der Abbildung durch gepunktete Linien dargestellt ist. Somit bildet das Bussystem die einzige Verbindung zwischen beiden Systemteilen, Nachrichten von Positionssensoren bzw. an Endstufen werden also über einen fremden Knoten hinweg an den eigentlichen Zielrechner weitergeleitet.

Die Zuordnung im Sinne einer wechselseitigen Kontrolle bleibt allerdings bestehen: ECU_2 überwacht weiterhin die Bewegung von Motor M_1 und passiviert diesen ggf. über die Endstufe O_1. Umgekehrt überwacht ECU_1 ebenfalls die Bewegung von M_2 und passiviert diesen ggf. über O_2, wie oben beschrieben. Der einzige Unterschied zur Systemvariante mit Dedizierter Redundanz besteht darin, auf welchem Weg die entsprechenden Informationen (d. h. Sensordaten und Passivierungsaufforderungen) übermittelt werden: An die Stelle direkter Kabelverbindungen wird die Weiterleitung durch eine fremde ECU hindurch und über das Bussystem hinweg gesetzt, wobei die Verwendung von Signaturen sicherstellt, dass Informationen auf diesem Weg nicht unerkannt verfälscht werden können. Dabei ist eine nicht rechtzeitig eintreffende Nachricht wie eine Passivierungsaufforderung zu werten.

Um nun begründen zu können, warum bei Verwendung dieses Verfahrens die Fail-Safe-Eigenschaft des Systems sichergestellt ist, ist zunächst ein Fehlermodell (vgl. Kapitel 2) aufzustellen. Die nachfolgende Tabelle 1 beinhaltet dementsprechend eine Aufstellung aller als möglich angenommenen Fehler des betrachteten Systems:

Tabelle 1: Fehlermodell Fail-safe-System (Entfernte Redundanz)

Komponente	**Fehlfunktion**
Motor	vollständig blockiert, reduzierte Geschwindigkeit
Sensor	beliebig
Kabel	Verfälschungen von Werten, aber keine unerkennbare Signaturerzeugung oder -verfälschung
Steuergerät (ECU)	Verfälschungen von Werten, aber keine unerkennbare Signaturerzeugung oder -verfälschung
Kommunikationskanal	Verfälschungen von Werten, aber keine unerkennbare Signaturerzeugung oder -verfälschung
Endstufe	unbeabsichtigte Passivierung oder Aktivierung
Brücke	Ansteuerung in die falsche Richtung (links, rechts, geradeaus), falls die zugehörige Endstufe Energie zur Verfügung stellt (d. h. nicht passiviert)
sonst. Mechanik	- (Perfektionskern)

Motoren können von sich aus keine Energie erzeugen und daher lediglich aufgrund von Reibung mit reduzierter Geschwindigkeit laufen bzw. vollständig blockiert sein. Sensoren können beliebig falsche Daten liefern oder komplett ausfallen. Alle Komponenten, die Informationen weiterleiten, dürfen die zugehörigen Signaturen nicht unerkannt verfälschen, können ansonsten aber beliebiges Fehlverhalten aufweisen. Endstufen dürfen den an sie angeschlossenen Motor beliebig aktivieren oder passivieren, während eine Brücke unabhängig von der durch das Steuergerät ausgegebenen Richtungsinformation im Fehlerfall eine Ansteuerung des angeschlossenen Motors in eine beliebige Richtung veranlassen kann, sofern ihr die Endstufe Energie zur Verfügung stellt (d. h. nicht passiviert). Weitere mechanische Bestandteile des Systems sind hier nicht Gegenstand der Betrachtung, lägen aber in jedem Fall im Perfektionskern und wären daher nicht von einer Fehlerbehandlung erfasst.

Für jeden der beschriebenen möglichen Fehlerfälle der Komponenten lässt sich nun zeigen, wie das System mit Entfernter Redundanz auf einen solchen Fehler reagiert und einen

sicheren Zustand, d. h. eine praktisch identische Position beider mechanischer Systemteile, garantiert. Im Einzelnen ergeben sich dabei die folgenden Situationen:

- Ein blockierter oder mit reduzierter Geschwindigkeit laufender Motor erzeugt eine Abweichung zwischen den von den Positionssensoren übermittelten Werten des oberen Teilsystems und den Positionswerten des unteren Teilsystems. Diese Abweichung wird von beiden, nach Annahme fehlerfreien Rechnern erkannt, woraufhin diese beide Motoren passivieren.
- Der Ausfall eines einzelnen Sensors (abweichender Wert, ungültige Signatur, kein Wert) führt nicht zur Passivierung, da in beiden Steuergeräten alle vier Sensorwerte vorliegen und somit eine Mehrheit für den richtigen Wert gebildet werden kann.
- Bei der Weiterleitung von Daten durch ein Kabel, den Kommunikationskanal oder ein Steuergerät kann die Signatur der übermittelten Daten zerstört werden. Sofern dies einen einzelnen Sensorwert betrifft, ist dieser Wert als fehlerhaft zu ignorieren. Nicht korrekt signierte Daten zur Aktivierung bzw. Passivierung werden von den Endstufen wie eine Passivierung gewertet.
- Ein dauerhaft fehlerhafter Kommunikationskanal würde dazu führen, dass generell keine gültigen Werte mehr übermittelt werden (Sollwerte, Istwerte, Signale zur Aktivierung bzw. Passivierung). Beide Rechner passivieren in einem solchen Fall nach Ablauf einer Zeitschranke die an sie angeschlossenen Motoren.
- Eine defekte Endstufe könnte den an sie angeschlossenen Motor beliebig aktivieren oder passivieren. Die Aktivierung eines nach Annahme fehlerfreien Motors bliebe ohne Folgen; eine Passivierung würde anhand abweichender Sensorwerte zwischen oberem und unterem Systemteil von beiden Rechnern erkannt und daher zur Passivierung beider Motoren führen.
- Eine defekte Brücke könnte zu einer falschen Ansteuerung eines Motors führen. Auch dies würde zu abweichenden Positionswerten zwischen oberem und unterem Teilsystem führen und somit eine Passivierung hervorrufen.
- Ein defekter Rechner kann den zugehörigen Motor ebenfalls falsch ansteuern. Die durch ihn durchgeleiteten Positionsdaten kann er jedoch aufgrund des Schutzes durch die Signatur nicht unerkennbar verfälschen. Die vom zweiten Rechner erkannte Abweichung würde daher zu einer Passivierung beider Motoren führen. Hierbei kann der fehlerhafte Rechner zwar ebenso die von ihm weiterzuleitende

Passivierungsaufforderung „unterschlagen“, ein Timeout-Mechanismus in der Endstufe (automatische Abschaltung, wenn nicht binnen einer bestimmten Zeit Aktivierungssignale von beiden Rechnern eingehen) kann jedoch auch in diesem Fall eine Abschaltung beider Motoren garantieren.

Betrachtet man nun ein konkretes Beispiel für die Anwendung von Fail-safe-Systemen, so wäre beispielsweise eine Ansteuerung von Landeklappen im Flugzeug denkbar. Hierbei wird gefordert, dass die Klappen im linken und rechten Flügel stets in annähernd identischer Position stehen, um die Flugeigenschaften nicht zu gefährden. Es ist jedoch zulässig, dass beide Klappen in einer identischen, aber nicht gewünschten Position verbleiben, da bei entsprechend langer Landebahn das Flugzeug auch in diesem Fall sicher gelandet werden kann. Das oben beschriebene Fail-safe-System mit Entfernter Redundanz wäre dann wie in Abbildung 8 dargestellt zu entwerfen:

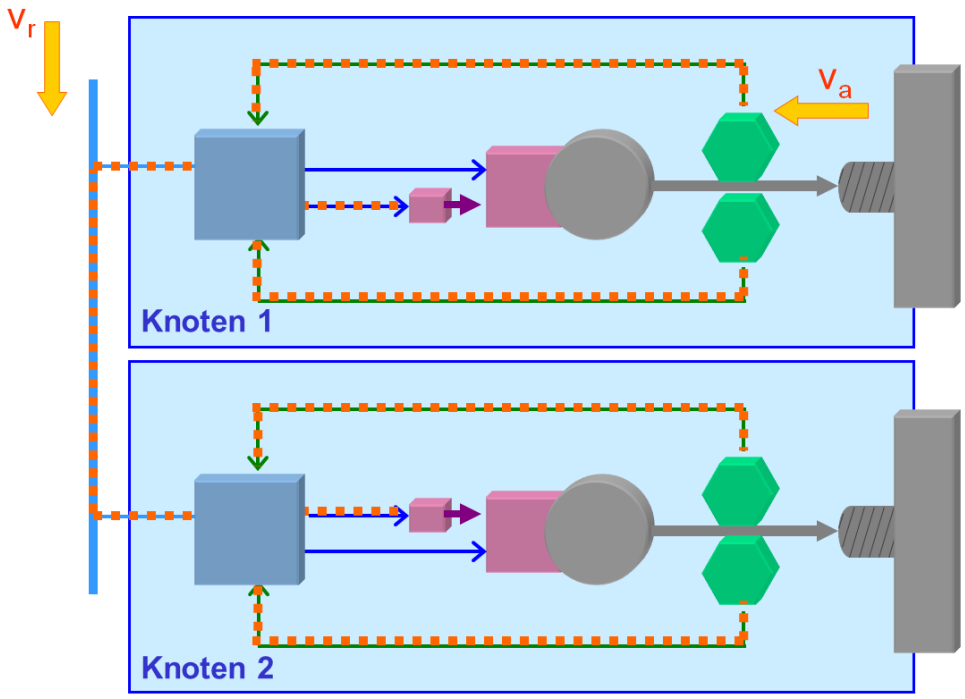


Abbildung 8: Fail-safe-Ansteuerung von Landeklappen, Variante A

Mit Blick auf den beschriebenen Anwendungsfall wird hier vor allem der Nutzen der wegfallenden Überkreuzverkabelung offensichtlich: Das durch jeden einzelnen Meter Kabel gegebene Gewicht verursacht über geschätzte 30 Betriebsjahre eines Verkehrsflugzeugs hinweg erhebliche Treibstoffkosten. Entfernte Redundanz erlaubt es, die Klappen an beiden Flügeln fehlertolerant anzusteuern, benötigt aber lediglich ein einkanaliges Bussystem als

Verbindung zwischen beiden Teilsystemen. Der Kabelbaum eines herkömmlichen Entwurfs mit Dedizierter Redundanz (Abbildung 6) müsste hingegen weit umfangreicher sein, um ein Fail-safe-Verhalten erreichen zu können.

Es wäre ebenso denkbar, dass für einen anderen Flugzeugtyp eine durchgehende Klappe verwendet wird, die aufgrund ihrer Größe von zwei Motoren angesteuert werden muss. Diese dürften ebenfalls nicht über einen längeren Zeitraum hinweg in unterschiedliche Richtungen angesteuert werden, da die Mechanik ansonsten Schaden nehmen würde. Die Fail-safe-Eigenschaft wird also weiterhin gefordert und kann auch hier mit Entfernter Redundanz erbracht werden, wie in Abbildung 9 dargestellt wird. Im Unterschied zur vorhergehenden Variante können in diesem System zwei der Positionssensoren wegfallen, wenn die Landeklappe in sich starr ist.

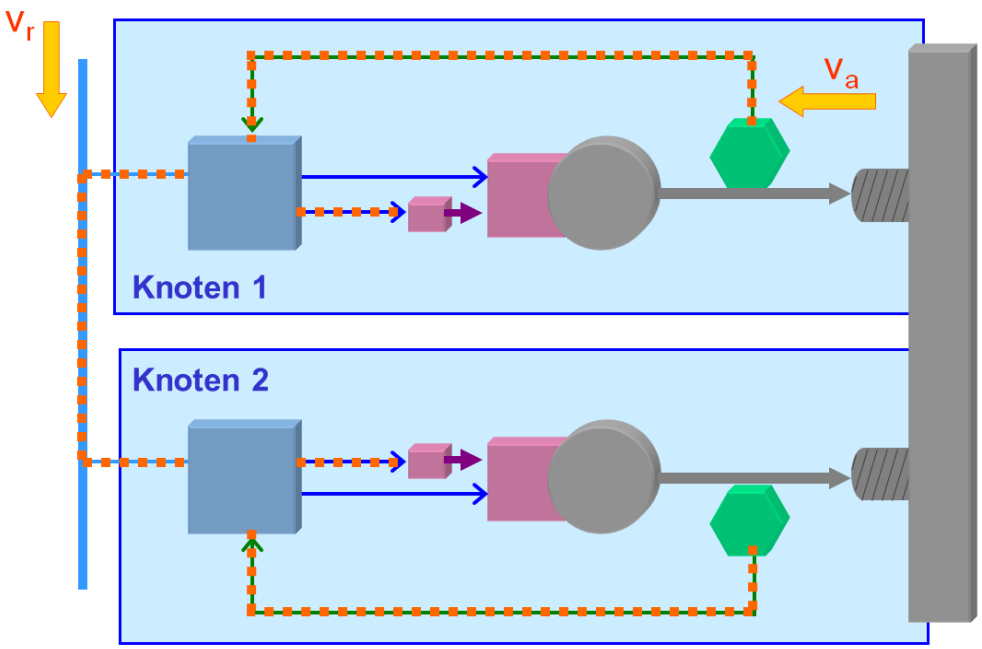


Abbildung 9: Fail-safe-Ansteuerung von Landeklappen, Variante B

Sofern die durchgehende Klappe klein genug ist (bzw. der verwendete Motor stark genug ist), kann noch eine weitere Variante Entfernter Redundanz eingesetzt werden (s. Abbildung 10):

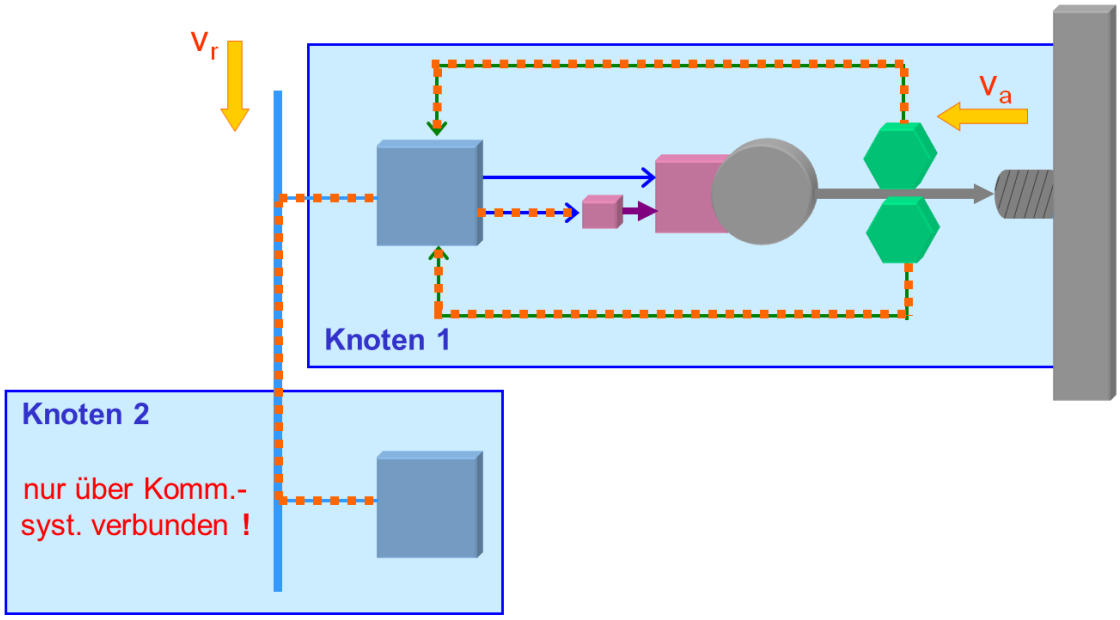


Abbildung 10: Fail-safe-Ansteuerung von Landeklappen, Variante C

Im hier dargestellten Fall benötigt der zweite Knoten keine direkte Verbindung mit der Sensorik/Aktuatorik mehr, ist jedoch für Fehlertoleranzzwecke weiterhin selbst nicht verzichtbar. Das entsprechende Steuergerät erhält Sensordaten in signierter Form über das Bussystem und sendet eigene Aktivierungs- bzw. (im Falle einer Abweichung) Passivierungsaufforderungen ebenfalls signiert über den Bus an die Endstufe des nunmehr einzigen Motors. Da das Bussystem die einzige Verbindung darstellt, kann dieser Knoten nun an einer beliebigen Stelle im Netzwerk platziert werden. Sofern auf bereits vorhandenen Knoten noch freie Rechenkapazität existiert, kann seine Funktionalität sogar ausschließlich in Software realisiert werden, so dass die für den Fail-safe-Betrieb notwendige Redundanz praktisch kostenlos ist.

4.3.3 Fail-operational-Konfiguration

Nicht für alle Systeme existiert ein sicherer Zustand ohne Erbringung der Nutzfunktion. So ist beispielsweise bezogen auf das Steuerruder eines Schiffes keine Position sicher, auch nicht die Geradeausfahrt, wenn ein Hindernis direkt voraus liegt oder sich gar nähert. Es tut sich somit eine Klasse fehlertoleranter Systeme auf, bei denen Sicherheit nur gemeinsam mit Zuverlässigkeit zu erreichen ist (vgl. erneut Kapitel 2). Von einer solchen, als Fail-operational-System bezeichneten Steuerung wird demnach gefordert, dass sie ihre Nutzfunktion auch beim Auftreten von Fehlern erbringt. Eine sehr typische Anwendungsklasse

sind dabei Systeme mit so genannter Einfehlertoleranz, welche also zu jeder gegebenen Zeit genau einen Fehler tolerieren.

Um das Beispielsystem aus Kapitel 4.3.1 zu einem solchen Fail-operational-System zu erweitern, ist erwartungsgemäß ein höherer Grad an Redundanz notwendig. Bevor für diese Anwendungsklasse die Verwendungsmöglichkeit Entfernter Redundanz dargestellt wird, soll zunächst auch hier der bisherige Stand der Technik in Form eines Systems mit Dedizierter Redundanz (s. Abbildung 11) erläutert werden.

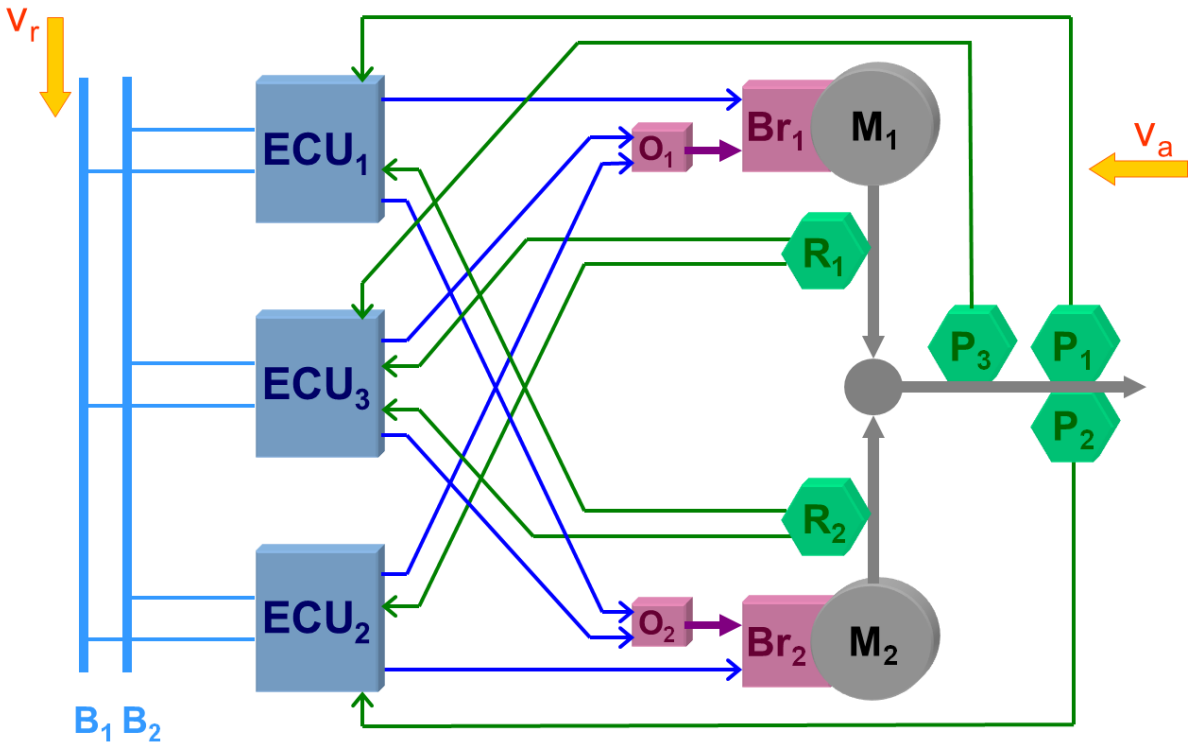


Abbildung 11: Fail-operational-Konfiguration, Dedizierte Redundanz

Das oben abgebildete System besteht aus insgesamt drei ECUs, welche über ein Doppelbussystem Sollvorgaben erhalten. Zwei der Rechner steuern jeweils über eine Brücke einen Motor an. Die Umdrehung beider Motoren wird über ein Differential gekoppelt, so dass sich die Ausgangswelle des Systems noch mit halber Geschwindigkeit bewegt, wenn einer der beiden Motoren stillsteht. Alle drei Knoten verfügen über einen Positionssensor, welcher durch Lieferung des Istwerts den Regelkreis schließt. Der mittlere Rechner (ECU_3) verwendet diese Information ausschließlich zu Passivierungszwecken: Analog zu ECU_1 und ECU_2 wird aus der Regeldifferenz zunächst ein Steuerbefehl berechnet. Dieser dient jedoch nicht zur Ansteuerung eines Motors, sondern wird verglichen mit der tatsächlichen Rotation der Motoren

(gemessen von den beiden Rotationssensoren R_1 und R_2). Hintergrund ist die Tatsache, dass gegenläufige Bewegungen der beiden Motoren am Differential zu einem Stillstand der Ausgangswelle führen würden. Um genau dies zu verhindern, ist sicherzustellen, dass fehlerhafte Motoren als solche erkannt und passiviert werden. Auszuschließen ist allerdings ebenso, dass ein fehlerhafter Rechner beide Motoren passiviert und hierdurch ebenfalls einen Systemstillstand herbeiführen würde. Aus diesem Grund sind die beiden Motoren jeweils von zwei Rechnern gemeinsam zu passivieren: es sind dies stets ECU_3 und diejenige ECU, welche den Motor nicht ansteuert (ECU_2 für Motor M_1 und ECU_1 für Motor M_2). Auch hier entsteht also die für fehlertolerante Systeme typische Überkreuzverkabelung zur Ermöglichung einer wechselseitigen Kontrolle einzelner Komponenten.

Auch für Fail-operational-Systeme ergibt sich somit bei Verwendung Dedizierter Redundanz nicht nur ein Mehraufwand bezüglich der notwendigen Redundanz selbst, sondern auch eine deutlich komplexere Systemstruktur im Vergleich zum nichtredundanten System und eine strukturelle Abhängigkeit der einzelnen Systemteile voneinander.

Auch für diese Klasse von Systemen soll Entfernte Redundanz die oben geschilderten Nachteile reduzieren, zugleich jedoch identische Fehlertoleranzeigenschaften im Vergleich zu herkömmlich aufgebauten Systemen gewährleisten. Wie im bereits im vorangegangenen Kapitel beschriebenen Fall des Fail-safe-Systems soll also auch hier nicht etwa ein höherer Grad an Fehlertoleranz erzielt werden, sondern eine als gegeben angenommene Fehlertoleranzanforderung mit möglichst geringen Kosten erreicht werden. Wie dies für ein Fail-operational-System gelingen kann, ist in nachfolgender Abbildung 12 dargestellt.

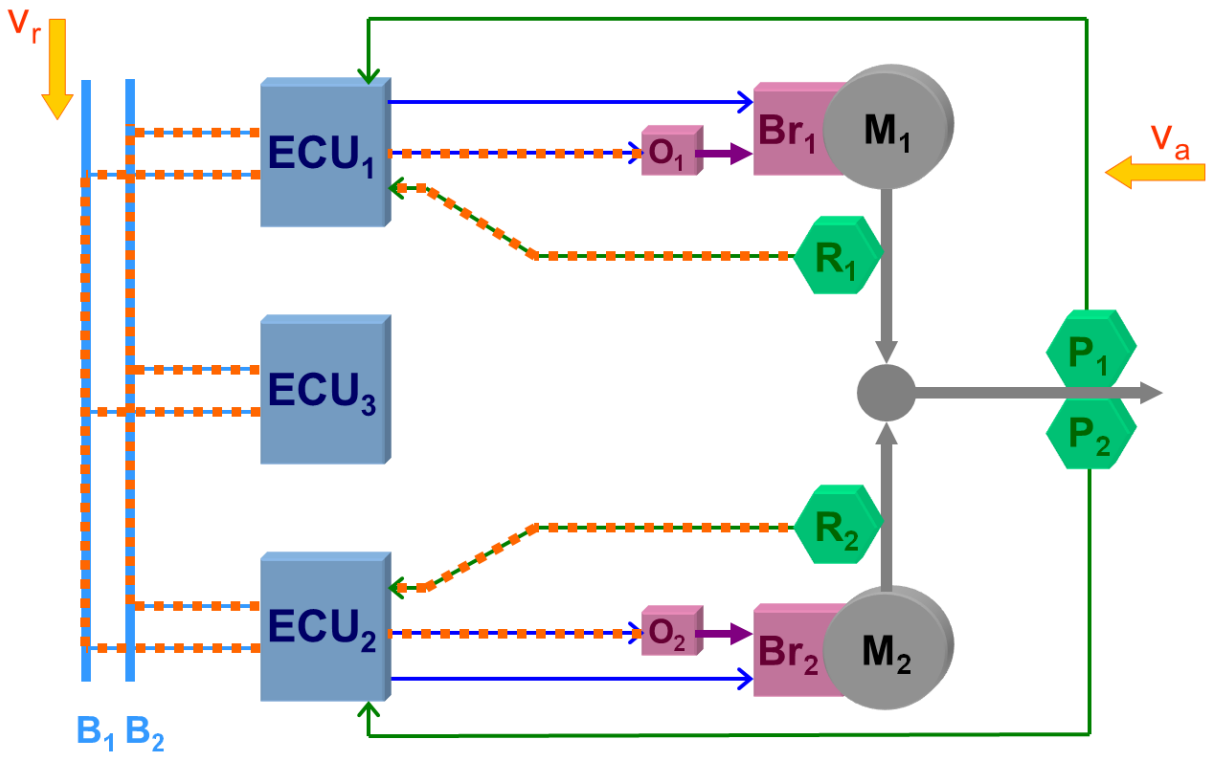


Abbildung 12: Fail-operational-Konfiguration, Entfernte Redundanz

Die Funktion des oben abgebildeten Gesamtsystems mit seinen einzelnen Komponenten unterscheidet sich zunächst nicht wesentlich vom bereits beschriebenen System mit Dedizierter Redundanz. Augenfällig ist jedoch, dass im Unterschied zu diesem die drei Systemteile nicht mehr „über Kreuz“ miteinander verbunden sind. Stattdessen werden die von den Rotationssensoren gelieferten Werte vom jeweils nächstgelegenen Rechner an die eigentlichen Zielrechner weitergeleitet: der Wert von R_1 wird über ECU_1 an ECU_2 und ECU_3 übermittelt, der Wert von R_2 über ECU_2 an ECU_1 und ECU_3. Erneut sind es ECU_3 und die den betreffenden Motor nicht ansteuernde ECU (also ECU_2 für Motor M_1 und ECU_1 für Motor M_2), welche gemeinsam für eine Passivierung zuständig sind. Die von den ECUs ausgehenden Passivierungs- bzw. Aktivierungsaufforderungen werden signaturgeschützt über den Bus und dazwischenliegende fremde Knoten hinweg an die betreffende Endstufe weitergeleitet und dort verarbeitet. Eine zusätzliche Veränderung gegenüber dem System mit Dedizierter Redundanz ergibt sich in Bezug auf die Positionssensoren: Durch die Verwendung des Zeitintegrals über R_1 und R_2 als Näherungswert kann ECU_3 auf einen eigenen Positionssensor verzichten, der Positionssensor P_3 kann folglich entfallen.

Unter Beibehaltung des bereits bekannten Fehlermodells aus Tabelle 1 kann nun auch für dieses System bzgl. jedes einzelnen Fehlerfalls begründet werden, warum das abgebildete System die Forderung nach Einfehlertoleranz erfüllt:

- Ein blockierender Motor hat keinen negativen Einfluss auf das System, sofern die Leistung des dann verbleibenden Motors erhöht werden kann oder von vorneherein groß genug ist um auch mit halber Kraft die geforderte Wirkung zu erzielen. Eine zusätzliche Passivierung ist nicht notwendig.
- Der Ausfall eines Positionssensors kann zu fehlerhaften Messwerten führen. Diese bewirken u. U., dass die zugehörige ECU eine Ansteuerung in die falsche Richtung veranlasst, was eine Passivierung des betreffenden Motors erfordert. Die Passivierung gelingt jedoch, da die beiden anderen ECUs (durch den eigenen Positionssensor bzw. unter Zuhilfenahme von R_1 und R_2) zu einer richtigen Einschätzung der Position gelangen können. Damit ist ihnen der eigentlich korrekte Steuerbefehl bekannt und sie können anhand des nach Annahme fehlerfreien Rotationssensors am falsch angesteuerten Motor eine korrekte Passivierungsentscheidung treffen. Da der zweite Motor richtig angesteuert wird, erbringt das Gesamtsystem weiterhin seine Funktion.
- Ein fehlerhafter Rotationssensor kann suggerieren, dass sich der entsprechende Motor in die falsche Richtung dreht. Dies würde die anderen beiden Knoten zu einer Passivierung veranlassen, welche jedoch aufgrund des noch funktionsfähigen zweiten Motors unkritisch ist.
- Eine unzulässige Passivierung durch eine defekte Endstufe würde den entsprechenden Motor abschalten. Der andere Motor ist jedoch hiervon nicht beeinträchtigt, so dass das Gesamtsystem seine Funktion weiterhin erbringt. Eine permanente Aktivierung bleibt gänzlich ohne Folgen, da alle anderen Komponenten nach Annahme fehlerfrei sind.
- Bei der Weiterleitung von Daten durch ein Kabel, einen Rechner oder einen Buskanal kann die zugehörige Signatur zerstört werden. Bei Sensorwerten liegen dann keine verwertbaren Informationen über den Zustand des entsprechenden Motors vor, was zu einer Passivierung führt. Bei zerstörten Aktivierungs- bzw. Passivierungsaufforderungen entscheiden sich die Endstufen wie folgt: Ein Motor bleibt aktiviert, wenn mindestens ein Knoten mit gültiger Signatur dafür stimmt. Gemäß Einfehlerannahme kann ausgeschlossen werden, dass sich der Motor falsch

bewegt (erster Fehler) und zugleich eine ECU seine Aktivierung fordert (zweiter Fehler).

- Fällt ein Buskanal komplett aus, so kann er ersetzt werden, indem auf dem anderen Buskanal identische Informationen transportiert werden. Es existieren Bussysteme für den industriellen Einsatz[8], welche dies gewährleisten.
- Eine fehlerhafte Brückenschaltung kann bewirken, dass ein Motor in die falsche Richtung angesteuert wird. Dies würde von beiden diesen Motor überwachenden Rechnern anhand eines Vergleichs der eigenen Steuerbefehle mit der tatsächlichen Rotation dieses Motors erkannt werden und zu einer Passivierung des betreffenden Motors führen. Der andere Motor arbeitet nach Annahme korrekt, so dass das Gesamtsystem weiterhin seine Nutzfunktion erbringt.
- Auch ein fehlerhafter Rechner könnte einen Motor in die falsche Richtung ansteuern. Er könnte zudem den an die anderen beiden, ihn überwachenden Knoten gelieferten Rotationswert zerstören (ihn aber aufgrund der Signatur nicht fälschen). Die von den beiden anderen Knoten in diesem Fall ausgesandten Passivierungsaufforderungen kann er ebenso zerstören. Durch einen Timeout-Mechanismus in der zugehörigen Endstufe, welche auf das Ausbleiben expliziter Aktivierungsaufforderungen reagiert, kann dies jedoch erkannt werden, so dass schließlich dennoch eine Passivierung erfolgt. Der fehlerfreie zweite Motor gewährleistet weiterhin ein Funktionieren des Gesamtsystems.
- Ein fehlerhafter Rechner könnte ebenso versuchen, beide Motoren zu passivieren. Da hierzu jedoch stets noch die Stimme eines weiteren Rechners notwendig ist, kann ihm dies bei Zugrundelegung einer Einfehlerannahme nicht gelingen.

Eine Beispielanwendung für ein solches fehlertolerantes System könnte etwa eine elektronisch geregelte Lenkung sein, wie sie in Abbildung 13 dargestellt ist. Die Sollvorgabe ist in diesem Fall der vom Fahrer durch den Einschlag am Lenkrad angegebene Lenkwinkel. Dieser wird elektronisch an zwei Stellmotoren übermittelt, welche unter Berücksichtigung der momentanen Position die Spurstange entsprechend bewegen, bis der gewünschte Winkel erreicht ist.

[8] Ein Beispiel hierfür ist das FlexRay-Datenbussystem.

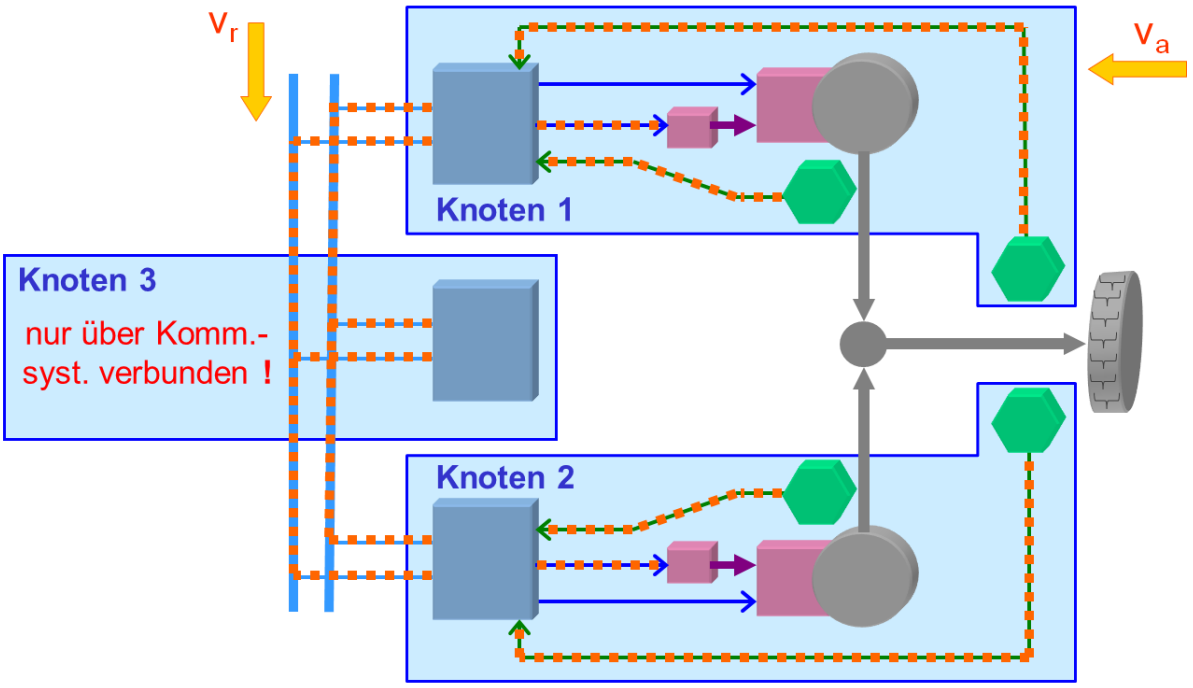


Abbildung 13: Fail-operational-Ansteuerung einer elektronischen Lenkung

Wie die vorangegangenen Abschnitte gezeigt haben, können auch für Fail-operational-Systeme die geforderten Fehlertoleranzeigenschaften ohne Einbußen mit Hilfe Entfernter Redundanz erbracht werden. Auch hier kann einer der Rechner durch die komplette Abkopplung von der Peripherie nunmehr beliebig im Netz platziert werden, oder, bei entsprechend vorhandenen Ressourcen, durch Software auf einem bereits existierenden fremden Rechner abgebildet werden. Ähnlich wie beim bereits betrachteten Fail-safe-System ist im Vergleich mit den Systemvarianten, welche unter Verwendung Dedizierter Redundanz entstanden, sichtbar, dass die Verkabelung „über Kreuz“ vollständig entfallen kann, wenn Daten signaturgeschützt durch fremde Knoten hindurchgeleitet werden. Hierdurch werden insgesamt weniger Kabel benötigt, das Gesamtsystem ist aber auch modularer aufgebaut, da die strukturelle Abhängigkeit zwischen den einzelnen Knoten aufgelöst wird.

Aufgrund der großen Bedeutung der Klasse fehlertoleranter Systeme mit Einfehlertoleranz dient das hier eingeführte, oben abgebildete System einer mit Entfernter Redundanz realisierten elektronischen Lenkung in Kapitel 5 der Arbeit in Form eines Fallbeispiels als Gegenstand einer umfassenden Untersuchung. Zunächst soll jedoch beschrieben werden, wie auch weitere Redundanzgrade mit Hilfe Entfernter Redundanz realisiert werden können.

4.3.4 Konfiguration zur Tolerierung von Doppelfehlern

Bislang wurde gezeigt, dass mit Entfernter Redundanz Systeme erstellt werden können, welche bezogen auf einen einzelnen Fehler ein Fail-safe- bzw. ein Fail-operational-Verhalten aufweisen. Das Konzept der Entfernten Redundanz ist jedoch nicht auf die Tolerierung von Einzelfehlern beschränkt. Abbildung 14 zeigt ein System mit Entfernter Redundanz, welches geeignet ist, beliebige Doppelfehler zu tolerieren. Zwei möglicherweise fehlerhaften Knoten ist hier eine Mehrheit von drei stets fehlerfreien Knoten gegenüberzustellen (vgl. Kapitel 2), was insgesamt zu einem so genannten 3-von-5-System führt. Durch die Nutzung Entfernter Redundanz können jedoch erneut Knoten komplett von der Peripherie entkoppelt und somit potentiell durch Software ersetzt werden.

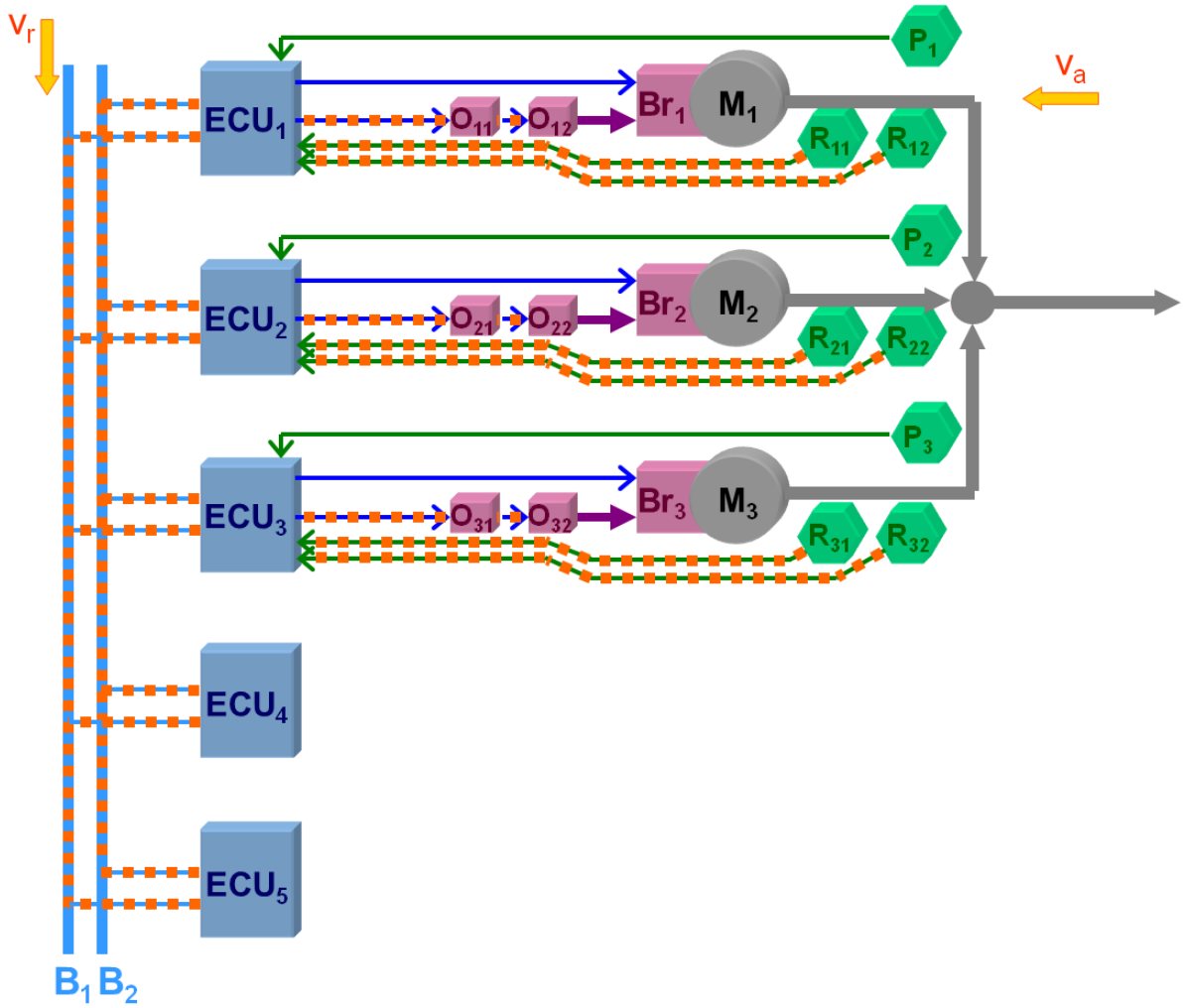


Abbildung 14: 3-von-5-System, Entfernte Redundanz

Das oben abgebildete System ist darauf ausgelegt, dass zu jedem Zeitpunkt zwei beliebige Fehler toleriert werden können. Hierdurch ergibt sich zunächst die Notwendigkeit, drei

Motoren M_1, M_2 und M_3 zu verbauen und ihre Ausgangswellen zu koppeln. Geht man davon aus, dass alle Motoren den gleichen Einfluss auf die insgesamt entstehende Bewegung haben, so kann diese auch beim Ausfall zweier Motoren noch mit einem Drittel der Kraft erfolgen.

Hierfür ist es jedoch notwendig, dass Motoren, die sich fehlerbedingt in die falsche Richtung drehen, angehalten werden. Dies wiederum erfordert pro Motor zwei Rotationssensoren und zwei Endstufen: Wenn ein Motor falsch angesteuert wird, könnte ein einzelner fehlerhafter Rotationssensor Daten liefern, welche für einen korrekten Betrieb des Motors sprächen und so eine Passivierung verhindern. Aufgrund der Zweifehlerannahme kann hingegen bei zwei Rotationssensoren höchstens einer von ihnen zusätzlich zum Motor defekt sein. Desgleichen würde eine einzelne defekte Endstufe verhindern können, dass ein sich in die falsche Richtung bewegender Motor passiviert wird. Zwei hintereinandergeschaltete Endstufen erlauben es hingegen, den zugehörigen Motor stets zu passivieren, wenn dieser fehlerhaft ist, da zusätzlich zum Motor höchstens eine von ihnen defekt sein kann.

Die Frage, wann eine Passivierung zulässig ist, steigt in ihrer Komplexität mit dem vorhandenen Redundanzgrad. Klar ist, dass ein Motor nur dann passiviert werden darf, wenn sich mindestens drei Knoten dafür entscheiden. Ansonsten könnten zwei Knoten selbst fehlerhaft sein (etwa ECU_1 und ECU_2) und ihre eigenen Motoren falsch ansteuern, zugleich aber Motor M_3 passivieren. Da dies nicht passieren darf, ist stets eine zusätzliche Stimme erforderlich. Damit nun aber ein fehlerhafter Motor tatsächlich passiviert wird, müssen die darüber befindenden Knoten fehlerfrei sein. Eine feste Zuordnung scheidet somit aus, da nicht im Voraus feststeht, welche Knoten fehlerfrei sind und welche fehlerhaft: Ließe man etwa Motor M_1 von ECU_2, ECU_4 und ECU_5 passivieren, Motor M_2 hingegen von ECU_1, ECU_4 und ECU_5, so würde, wenn sowohl ECU_1 als auch ECU_2 fehlerhaft, keiner der beiden Motoren passiviert. Um also zu garantieren, dass immer die Meinung von drei fehlerfreien Knoten berücksichtigt wird, ist eine „3-von-4-Entscheidung“ erforderlich: Ein Motor wird passiviert, wenn drei von vier der ihn nicht ansteuernden ECUs hierfür stimmen. Umgekehrt formuliert bleibt ein Motor aktiviert, sobald zwei der ihn nicht ansteuernden ECUs dies durch eine gültig signierte Aktivierungsaufforderung veranlassen: Da höchstens zwei Fehler zugleich vorliegen können, sind niemals beide Aktivierungsaufforderungen unrichtig und eine weitere Komponente (welche eine falsche Ansteuerung des Motors verursachen könnte) fehlerhaft.

Entsprechend dem Grundgedanken Entfernter Redundanz sind möglichst viele Knoten (in diesem Fall ECU_4 und ECU_5) ausschließlich über das Bussystem mit dem Rest des Systems verbunden. Insbesondere haben diese beiden Knoten keine eigenen Positionssensoren und berechnen stattdessen Näherungswerte aus den an sie weitergeleiteten Rotationsangaben.

Da für beide Knoten die Rotation aller Achsen in das Ergebnis eingehen muss, wäre es eine zulässige Möglichkeit, dass ECU_4 diesen Wert auf Basis von R_{11}, R_{21} und R_{31} errechnet, ECU_5 anhand von R_{12}, R_{22} und R_{32}. Im ungünstigen Fall zweier übereinstimmend falscher Positionssensoren kann es tatsächlich so sein, dass sich die beiden zugehörigen Motoren zunächst kurzzeitig in die falsche Richtung bewegen. Allerdings kennen durch die obige Konstruktion ECU_4 und ECU_5 die wahre Position (die Rotationssensoren sind nach Annahme fehlerfrei) und passivieren zusammen mit dem verbleibenden Rechner die auf der Grundlage fehlerhafter Positionssensoren falsch angesteuerten Motoren.

4.3.5 Allgemeines Schema

Aufbauend auf den bisherigen Ergebnissen wird in der nachfolgenden Abbildung 15 nun der allgemeine Fall eines n-von-m-Systems mit Entfernter Redundanz dargestellt.

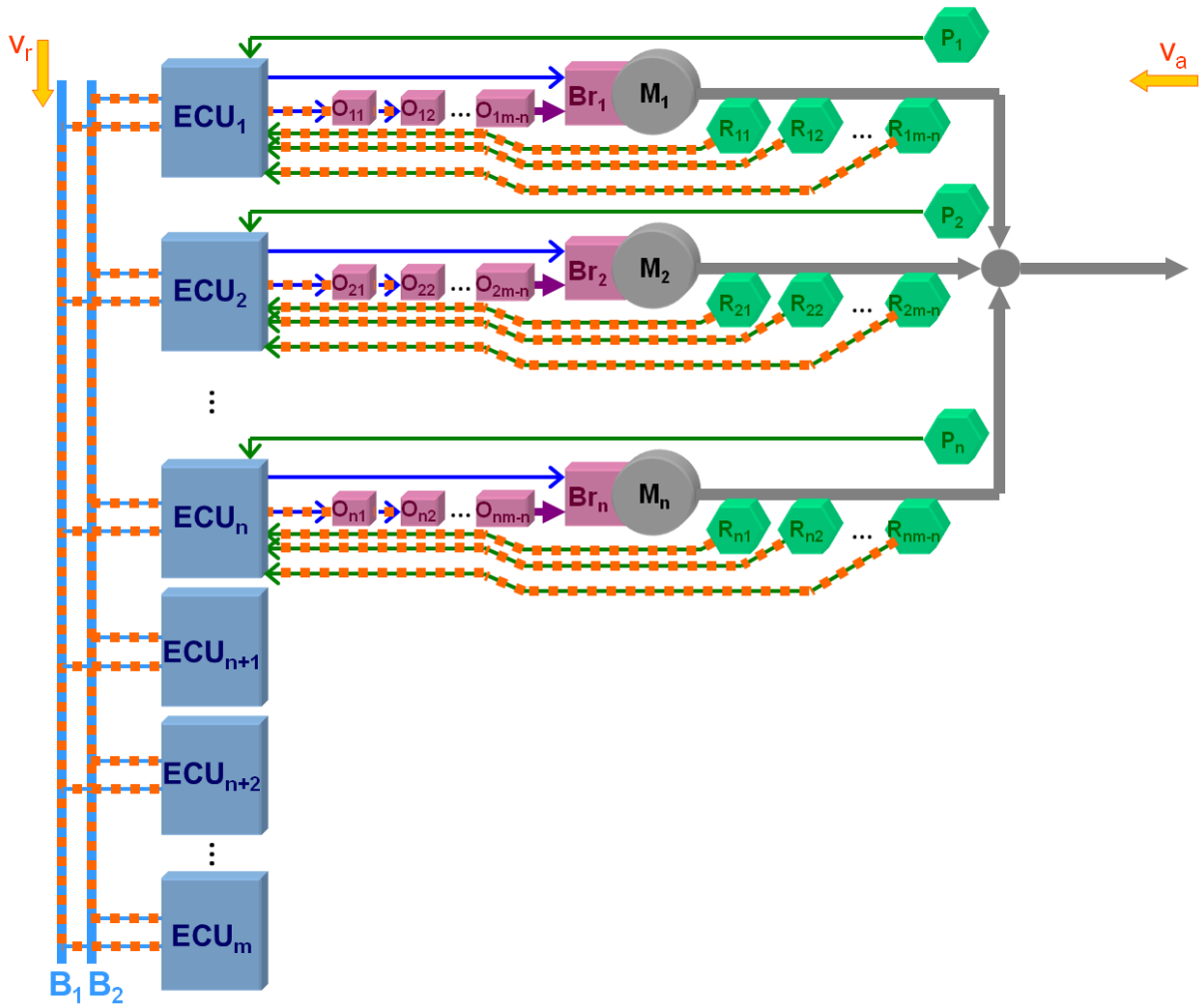


Abbildung 15: n-von-m-System, Entfernte Redundanz

Viele Überlegungen lassen sich unmittelbar aus den bereits beschriebenen Beispielen eines 2-von-3- sowie eines 3-von-5-Systemes ableiten. So sind im allgemeinen Fall n von m fehlerfreie Knoten notwendig, um $k = m - n$ Fehler tolerieren zu können. Es gilt, wie bereits in Kapitel 3 ausgeführt, $n = k + 1$ und somit $m = 2k + 1$.

Es ist ausreichend, n Knoten mit der Peripherie zu koppeln, da höchstens $n - 1$ Motoren fehlerhaft angesteuert werden können, womit noch ein richtig angesteuerter Motor verbleibt. Folglich können $m - n$ Knoten als Entfernte Redundanz beliebig im Netzwerk platziert werden. Jeder der einen Motor ansteuernden Knoten besitzt einen Positionssensor, um den Regelkreis zu schließen und $m - n$ Rotationssensoren zur Überwachung des Motors sowie $m - n$ Endstufen, um den Motor ggf. passivieren zu können. Da höchstens $m - n$ gleichzeitige Fehler angenommen werden, ist auf diese Weise ausgeschlossen, dass ein Motor und alle zugehörigen Endstufen oder ein Motor und alle zugehörigen Rotationssensoren zugleich fehlerhaft sind.

Passivierungsentscheidungen über einen Motor werden von allen Knoten getroffen, die diesen Motor nicht ansteuern (dies sind $m - 1$ Knoten). Jede Endstufe erhält diese Nachrichten und bleibt aktiviert, wenn $m - n$ Knoten mit gültiger Signatur dafür stimmen. Bei dieser Konstruktion müsste ein falsch angesteuerter Motor von $m - n$ fehlerhaften Knoten aktiviert werden, um dem System zu schaden. Da gemäß Annahme jedoch nur $m - n$ Fehler zugleich vorkommen können, ist es ausgeschlossen, dass, wie im beschriebenen Fall, $m - n + 1$ Fehler auftreten.

In folgender Abbildung 16 soll nunmehr das algorithmische Verhalten der Knoten im Detail beschrieben werden. Da das in Abbildung 15 vorgestellte allgemeine n-von-m-System zugrundegelegt wird, lässt sich der angegebene Pseudocode auch als ein grundsätzliches Schema für Entfernte Redundanz verstehen.

Zu beachten ist, dass hier das Verhalten *fehlerfreier* Knoten angegeben ist. Fehlerhafte Knoten können sich im Rahmen des Fehlermodells aus Tabelle 1 beliebig verhalten. Damit bereits zu Beginn vorliegende Fehler als solche erkannt werden können, sei der Ausgangspunkt stets ein sich im Stillstand befindliches System bei bekannter Initialposition. Ebenfalls bekannt sei der initiale Steuerbefehl aller fehlerfreier Knoten.

```
// Verhalten aller fehlerfreien Knoten k


// empfange Rotation jedes fremden Knotens i mit eigenem Aktuator
Rij := empfange R-Wert;    i ∈ {1,…,n},    j ∈ {1,…,m-n}

// berechne Akt.-/Pass.aufforderung für jeden fremden Knoten i mit eigenem Aktuator
Akij := (Ck=Ri,1=Ri,2=…=Ri,m-n);    i ∈ {1,…,n},    j ∈ {1,…,m-n}

// empfange Sollwert S
Sk := empfange Sollwert;

// ermittle Istwert P
if (k<=n) {  // für alle n Knoten mit eigenem Aktuator
     Pk := lies eigenen P-Sensor;    k ∈ {1, …, n}
} else {  // für alle m – n Knoten ohne eigenen Aktuator
     if (zyklus = 0) {  // im ersten Regelzyklus
          // verwende bekannten Initialwert
          Pk := P0;
     } else {  // in allen weiteren Zyklen
          // berechne Näherungswert aus Rotationssensoren
          Pk := Pk + (R1,k-n+R2,k-n+…+Rn,k-n);    k ∈ {n+1,…,m}
     }
}

// berechne neuen Steuerbefehl C
Ck := ermittle Vorzeichen(Sk – Pk);    k ∈ {1,…,m}
```

Abbildung 16: Allgemeines Schema Entfernter Redundanz (Pseudocode)

Der Zyklus beginnt für alle fehlerfreien Knoten k (Achtung, k ist hier ein Bezeichner und steht nicht für die Fehleranzahl!) mit dem Empfang der Rotationswerte R_{ij} von jedem fremden Knoten i mit eigenem Aktuator und jedem dort angebrachten Rotationssensor j (alle n Knoten mit eigenem Aktuator verfügen über jeweils $m - n$ Rotationssensoren).

Aus dem so vorliegenden Bild über die Bewegung der einzelnen Eingangswellen des Differentials lässt sich durch Abgleich mit dem zuletzt berechneten Steuerbefehl (bzw. dem bekannten Initialwert) C_k ermitteln, ob ein dem Knoten i zugeordneter Aktuator aus Sicht von Knoten k richtig angesteuert wird. Dies ist bei Übereinstimmung mit dem Steuerbefehl der Fall. Abgeleitet werden hieraus die Aktivierungs- bzw. Passivierungsaufforderungen A_{kij} für den fremden Knoten i. Hierbei ist jede diesem Knoten zugeordnete Endstufe j zu berücksichtigen (alle n Knoten mit eigenem Aktuator besitzen jeweils $m - n$ Endstufen).

Rotationswerte ohne gültige Signatur führen dabei stets zur Passivierung aller Endstufen des betreffenden Knotens.

Die Wirkung der Aktivierungs- bzw. Passivierungsauforderungen in den Endstufen stellt sich so dar, dass jede Endstufe genau dann aktiviert bleibt, wenn $n - 1$ der $m - 1$ fremden Knoten mit gültiger Signatur dafür stimmen (hiermit sind alle Fehlermöglichkeiten ausgeschöpft und der zugehörige Aktuator kann nicht ebenfalls fehlerhaft angesteuert werden).

Sodann empfängt jeder Knoten k (auch Knoten ohne eigenen Aktuator) die Sollvorgabe über das Bussystem. In Knoten mit eigenem Aktuator wird dieser Wert S_k auch zur Ansteuerung verwendet, in Knoten ohne eigenen Aktuator ausschließlich zur Überwachung fremder Knoten bzw. der durch sie angesteuerten Aktuatoren.

Die Berechnung des zugehörigen Istwertes P_k hängt davon ab, ob Knoten k einen eigenen Aktuator besitzt. Wenn dem so ist (also bei insgesamt n Knoten), kann der Wert des dann vorhandenen Positionssensors verwendet werden. Andernfalls ist mit Hilfe der Rotationssensoren ein Näherungswert zu berechnen. Für jeden der $m - n$ Knoten ohne eigenen Aktuator muss dabei die Rotation aller n Wellen unabhängig voneinander berücksichtigt werden. Dies gelingt, da an jedem Knoten mit eigenem Aktuator $m - n$ redundante Rotationssensoren angebracht sind. Zu beachten ist allerdings, dass für alle Motoren, die Knoten k selbst passiviert hat, fortan als Rotationswert „0“ zu verwenden ist.

Aus der Regeldifferenz zwischen der Sollvorgabe und dem nun vorliegenden Istwert berechnet jeder Knoten „seinen“ Steuerbefehl C_k. Wie bereits weiter oben beschrieben, dient dieser entweder auch zur Ansteuerung des eigenen Aktuators oder ausschließlich zur Überwachung fremder Knoten. Die Wirkung auf die Aktuatoren wird zusätzlich durch die ggf. passivierenden Endstufen beeinflusst, so dass im nächsten Zyklus für die tatsächliche Rotation $\underline{R_k}$ der dem Knoten k zugeordneten Welle und die tatsächliche Position $\underline{P}$ der angesteuerten Mechanik gilt:

- $\underline{R_k} = \underline{C_k} \cdot \underline{A_{k,1}} \cdot \underline{A_{k,2}} \cdot \ldots \cdot \underline{A_{k,m-n}}, \quad k \in \{1, \ldots, n\}$
- $\underline{P} = \underline{P} + (\underline{R_1} + \underline{R_2} + \cdots + \underline{R_n})$

Dabei ist $\underline{C_k}$ der tatsächliche Steuerbefehl (0, 1 oder – 1) des Knotens k und $\underline{A_{k,j}}$ der tatsächliche Aktivierungszustand der entsprechenden Endstufe (1 = aktiviert, 0 = deaktiviert). Vorauszusetzen ist, dass die Passivierung so schnell geschieht, dass sich eine kurzzeitig falsche Rotation wie „keine Rotation“ auswirkt.

4.4 Möglichkeiten zur Effizienz- und Nutzensteigerung

Wie in den vorangegangenen Abschnitten dargestellt, bietet Entfernte Redundanz die Möglichkeit, in erheblichem Umfang auf Hardware in Form von Steuergeräten und Kabeln zu verzichten, erzeugt jedoch einen erhöhten Kommunikationsbedarf bei der Weiterleitung von Daten über zentrale Bussysteme. Da Entfernte Redundanz als Konzept hinsichtlich der genauen Umsetzung generisch ist, stellt sich folglich die Frage nach möglichen Effizienz- und Nutzensteigerungen in diesem Bereich. Einige Überlegungen hierzu sollen im Folgenden kurz skizziert werden.

Zunächst ist es möglich, am Signaturverfahren selbst anzusetzen. Typischerweise stehen sich hier der durch ein Verfahren erreichbare Schutz und die Menge der zusätzlichen zu übertragenden Daten als konfliktäre Ziele gegenüber. So benötigen kryptografische Hash-Verfahren meist erheblich mehr an Ressourcen (als Beispiele seien MD5 mit 128 Bit sowie SHA-1 mit 160 Bit genannt) als einfache Verfahren wie übliche CRCs (so z. B. 32 Bit bei Ethernet). Gegenwärtige Überlegungen in potentiellen Anwendungsfeldern Entfernter Redundanz bewegen sich (je nach Konfiguration) sogar im Bereich von nur 8 Bit für ein Signaturverfahren (AUTOSAR, 2009). Das in Kapitel 4.2 vorgestellte Verfahren schreibt die Länge der Signatur nicht vor, wurde jedoch im Rahmen dieser Arbeit erfolgreich in einer 12 Bit-Variante verwendet, was bereits nahe am o. g. Wert liegt und bei geeigneter Parameterwahl den Schutz des dem Verfahren zugrundeliegenden CRCs bietet, zugleich jedoch die Authentizität von Nachrichten überprüfbar macht (vgl. Kapitel 5.2.4).

Auch in diesem unteren Anforderungsbereich gibt es jedoch Optimierungsspielraum. So könnte, wie in (Echtle & Kimmeskamp, 2009) vorgeschlagen, die zusätzlich zur Signatur verwendete Sequenznummer zerlegt werden, um bei gleichem Übertragungsaufwand einen größeren Zahlenbereich abzudecken. Dabei wird in jedem Zyklus ein anderer Ausschnitt aus der gesamten Sequenznummer übertragen, so dass diese erst nach einigen Zyklen vollständig vorliegt, dafür jedoch insgesamt eine erheblich größere Periode aufweisen kann. Ebenfalls in Kapitel 5.2.4 wird diese Möglichkeit im Detail untersucht. Insgesamt wird mit dort beschriebenen Verfahren bei vergleichbarem Aufwand ein weit größerer Zahlenbereich erreicht, als beispielsweise mit der gegenwärtig für AUTOSAR verfolgten Lösung eines 4-Bit-Zählers mit insgesamt 16 möglichen Sequenznummern.

Eine andere Möglichkeit besteht darin, die Eigenschaften der konkret verwendeten Sensoren bzw. Aktuatoren auszunutzen. Wenn diese selbst netzwerkfähig sind, entfällt mög-

licherweise die Notwendigkeit einer Weiterleitung der Messwerte bzw. Steuerbefehle über ECUs und der damit verbundene Aufwand.

Die ohnehin im Rahmen Entfernter Redundanz entstehende Informationsredundanz könnte zudem in weiteren Knoten genutzt werden. In den bislang beschriebenen Systemen wird etwa das Wissen um die Rotation einzelner Wellen nur insoweit genutzt, als die geforderten Fehlertoleranzeigenschaften erreicht werden, es könnte jedoch auch zu Optimierungszwecken in Bezug auf die Ansteuerung dienen. Die vorhandenen Informationen könnten sogar von anderen Systemen genutzt werden, insbesondere von sog. „Integrated Safety Systems“ (vgl. Kapitel 6.4). Dadurch sinkt zwar nicht der Aufwand, aber es wird ein zusätzlicher Nutzen geboten.

Die möglicherweise entstehenden Abweichungen bei der näherungsweisen Berechnung der Position über Rotationssensoren ist ein Ausgangspunkt für die Überlegung, dass Systemvarianten mit zusätzlichen Positionssensoren eine bessere Genauigkeit erzielen könnten. Diese Lösung wäre mit einer Kostensteigerung verbunden, die von der Anwendung und ihren genauen Anforderungen abhängt, könnte jedoch ebenfalls einen zusätzlichen Nutzen darstellen.

5 Analyse am Fallbeispiel einer elektronischen Lenkung

5.1 Vorbemerkungen und Auswahl geeigneter Modellierungsmethoden

Anhand eines durchgehenden Fallbeispiels soll nun in den folgenden Abschnitten demonstriert werden, dass das Konzept der Entfernten Redundanz nicht nur aus Fehlertoleranzsicht geeignet ist, sondern auch in der Praxis erfolgreich implementiert werden kann. Als Anwendungsbeispiel wird hierfür die in Kapitel 4.3.3 bereits beschriebene Konfiguration mit Einfehlertoleranz für eine elektronisch geregelte Lenkung verwendet (s. Abbildung 17).

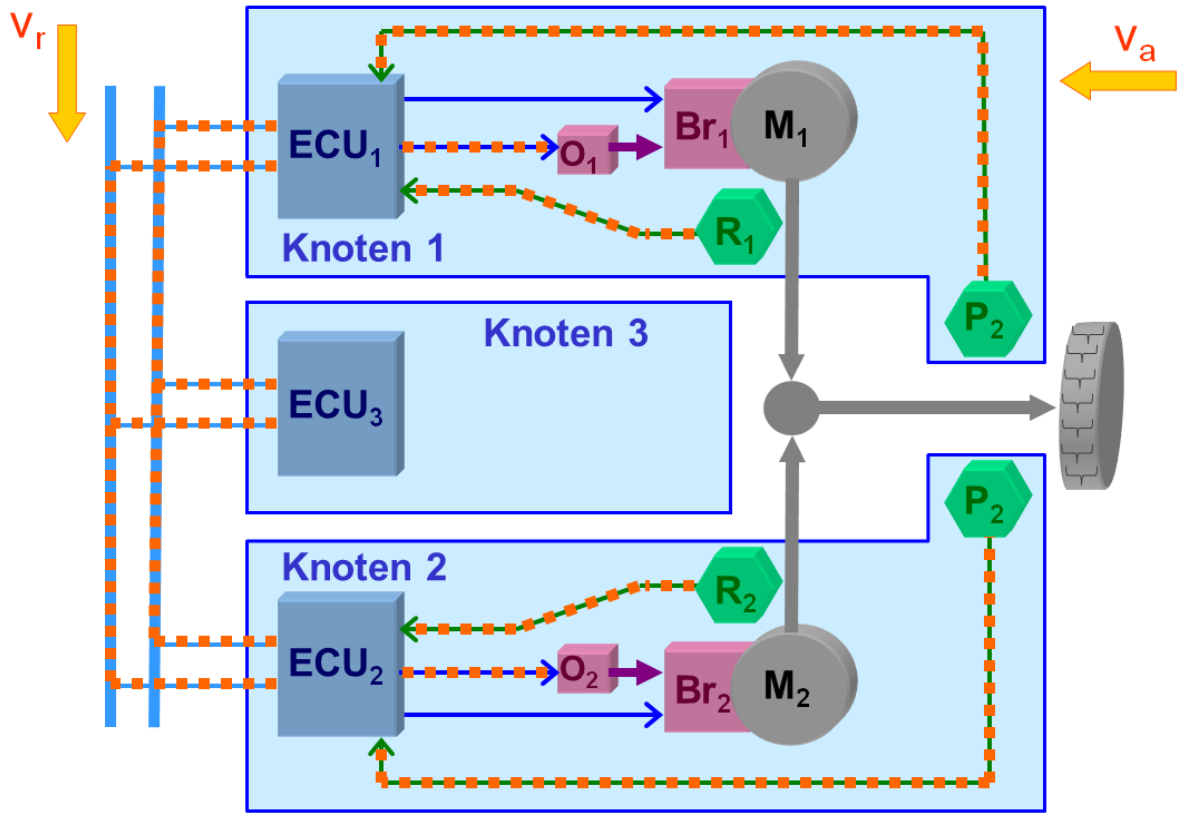


Abbildung 17: Elektronisch geregelte Lenkung

Zwei redundante Motoren wirken in diesem System über ein Differential auf ein Lenkgetriebe ein, welches den Rechts-Links-Einschlag der Vorderräder eines Automobils ermöglicht. Die beiden Motoren M_1 und M_2 werden je über ein Steuergerät (ECU_1 bzw. ECU_2) angesteuert. Zusammen mit der erneut ausschließlich zu Überwachungszwecken dienenden ECU_3 bilden diese ein System mit Entfernter Redundanz: Sensordaten sowie Aktivierungs- und Passivierungsaufforderungen gelangen nicht mittels direkter Kabelverbindungen, sondern durch Weiterleitung über fremde Steuergeräte und das Bussystem hinweg an ihr Ziel.

Für eine Analyse des oben abgebildeten Beispielsystems mit Entfernter Redundanz werden in der vorliegenden Arbeit verschiedene Modellierungsmethoden verwendet, welche es erlauben, jeweils unterschiedliche Aspekte detailliert zu betrachten, im Ganzen jedoch einen breiten Überblick über das Untersuchungsobjekt zu geben. Die folgende Abbildung 18 zeigt in Anlehnung an ein sog. Y-Diagramm (Gajski & Kuhn, 1983), wie die verwendeten Methoden „Fehlerbaumanalyse“, „Formale Verifikation“ „Funktionale Simulation“, und „Prototypische Implementierung“ eine Vertiefung in die Untersuchungsbereiche „Systemstruktur“, „Algorithmisches Verhalten“ und „Technische Realisierung“ ermöglichen. Der graue Pfeil gibt dabei die Reihenfolge an, in der die Untersuchungen im Rahmen der Arbeit durchgeführt wurden und in der sie im Folgenden auch dargestellt werden.

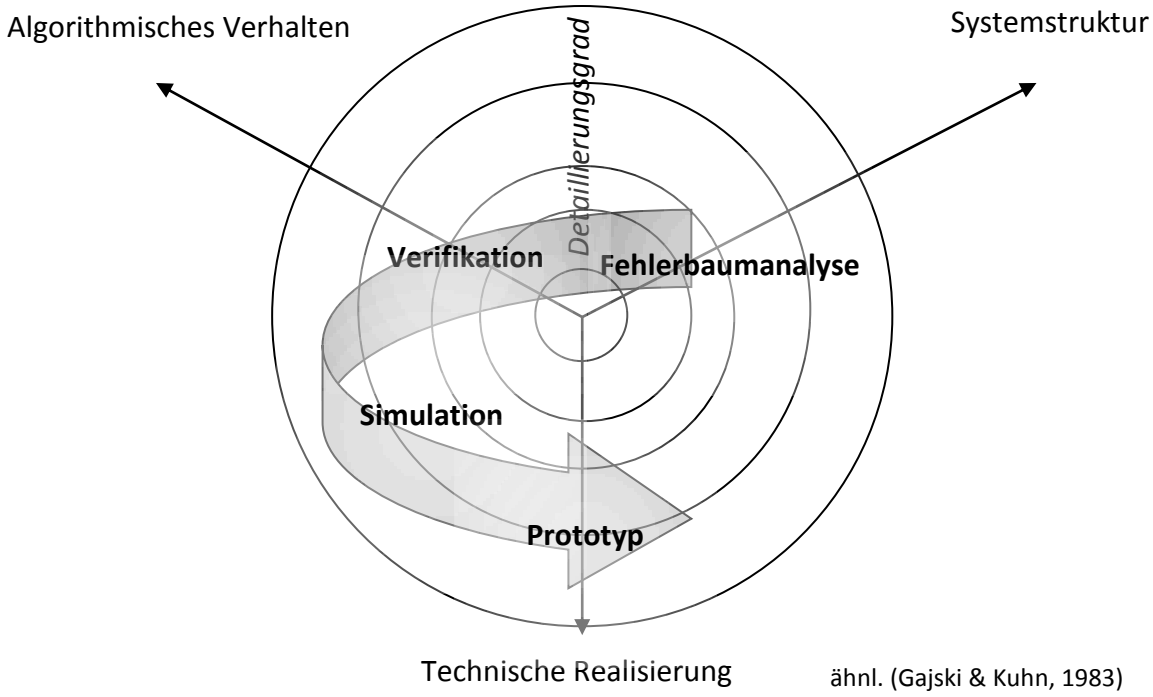


Abbildung 18: Modellierungsarten und -ziele

Zunächst wird das Beispielsystem mit Entfernter Redundanz anhand einer **Fehlerbaumanalyse** untersucht. Diese betrifft den grundsätzlichen strukturellen Aufbau des Systems und ermöglicht nicht nur eine Aussage über qualitative Fehlertoleranzeigenschaften, sondern bei Zugrundelegung entsprechender Annahmen über einzelne Komponenten auch quantitative Aussagen über die erreichbare Zuverlässigkeit des Systems. Von besonderem Interesse ist dabei ein Vergleich zur Dedizierten Redundanz als bisherigem Stand der Technik.

Die Erkenntnisse in Bezug auf das System mit Entfernter Redundanz sollen sodann durch weitere Untersuchungen vertieft werden. Hierzu dient ein immer noch relativ abstraktes Modell des Beispielsystems, welches sich jedoch aufgrund eben dieser Eigenschaft **formal verifizieren** lässt. Dieses Modell bildet erstmals die Funktion der einzelnen Systemkomponenten in vereinfachter Form ab und ermöglicht es durch Fehlerinjektion, beweisbare Aussagen über das Systemverhalten im Fehlerfall zu treffen.

Im dritten Schritt zeigt eine **funktionale Simulation** das Verhalten des Systems detaillierter. Hierbei wird nicht nur die Funktion der physikalischen Komponenten nachgebildet, sondern erstmals auch das Bussystem als Bestandteil des Modells vollständig simuliert. Auch in diesem Modell sind Fehlerinjektionsexperimente möglich, welche allerdings aufgrund des höheren Detaillierungsgrads nun auf einzelne Situationen beschränkt bleiben müssen. Sie ermöglichen es jedoch, das Systemverhalten unter realistischen Rahmenbedingungen kleinschrittig nachzuvollziehen.

Schließlich soll anhand einer **Hardwareimplementierung** und mit Hilfe eines realen Bussystems gezeigt werden, dass sich die für Entfernte Redundanz benötigten Funktionen in Hardware realisieren lassen und u. a. bezüglich ihres Zeitverhaltens in Rahmen des Gesamtsystems verwendbar sind. In diesem letzten Schritt erfolgt die vollständige Umsetzung der Verfahren zur Erzeugung bzw. Prüfung von Signaturen und Sequenznummern in einer programmierbaren Schaltung einschließlich ihrer Analyse unter Fehlerinjektion.

Neben der möglichst vollständigen Abdeckung der drei Untersuchungsbereiche „Systemstruktur“, „Algorithmisches Verhalten“ und „Technische Realisierung“ ist auch der Abstraktionsgrad der verwendeten Modelle in Bezug auf die durch sie gewonnenen Aussagen von Bedeutung. Die im Rahmen der vorliegenden Arbeit entstandenen Modelle weisen bewusst unterschiedliche Detaillierungsgrade auf, um sowohl die generelle Eignung des Konzepts demonstrieren, als auch konkrete Ausgestaltungsmöglichkeiten bzgl. seiner Umsetzung untersuchen zu können. Anfangs sind daher zwar formal beweisbare, aber relativ abstrakte Aussagen möglich, später konkretere Empfehlungen, die jedoch hierfür auf den jeweils betrachteten Anwendungsfall beschränkt sind.

Die Darstellungen aller vier in den folgenden Kapiteln betrachteten Teilmodelle bestehen in einheitlicher Form jeweils aus einer kurzen Einführung in die verwendete Methode, einer Beschreibung des erstellten Modells und einer abschließenden Bewertung der durch seine Analyse gewonnen Erkenntnisse.

5.2 Durchführung der Untersuchung

5.2.1 Fehlerbaumanalyse

5.2.1.1 Methode

Ein **Fehlerbaum** *(engl.: fault tree)* zeigt in grafischer Form, auf welche Weise Komponentenfehler zum Totalausfall des betrachteten Systems führen können (Schneeweiss, 1999). Er ist damit das Abbild einer booleschen Funktion

$$X_S = \varphi(X_1, \dots, X_n) \ ,$$

wobei X_S den Systemausfall beschreibt und X_i die Fehler-Indikatorvariablen der einzelnen Komponenten sind (ebd.). Zu beachten ist allerdings, dass ein solcher Fehlerbaum nicht immer vollständig sein muss und es aus praktischen Gründen oftmals auch nicht sein kann. Somit liegt es in der Verantwortung desjenigen, der den Baum erstellt, zu gewährleisten, dass keine relevanten Zusammenhänge unbeachtet bleiben. Die Aufstellung des Baumes erfolgt dabei stets „top-down“, also deduktiv vom Systemausfall ausgehend bis hin zu seinen nicht weiter sinnvoll zerlegbaren Ursachen.

Fehlerbäume können aus einer Vielzahl unterschiedlicher Elemente bestehen. Für eine entsprechende Übersicht sei auf (Haasl, Roberts, Vesely, & Goldberg, 1981) verwiesen. Für die vorliegende Arbeit werden lediglich die in Abbildung 19 dargestellten Symbole benötigt:

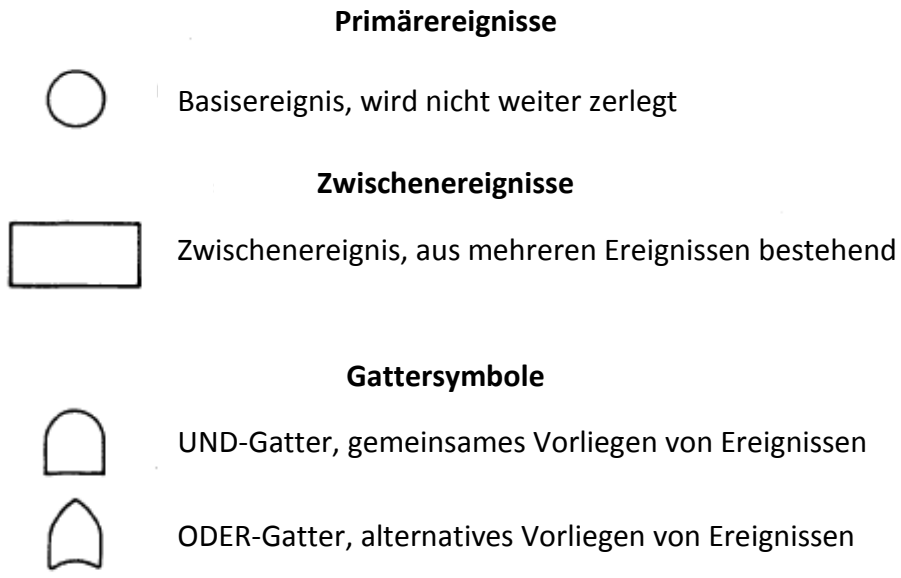


Abbildung 19: In Fehlerbäumen verwendete Symbole (Auswahl)

Die Semantik boolescher Funktionen wird in Fehlerbäumen dargestellt, indem für Fehler ursächliche Ereignisse durch logische Verknüpfungen miteinander in Beziehung gebracht werden. Die Blätter des dabei entstehenden Baumes sind stets sog. Primärereignisse, welche nicht weiter zerlegt werden. Alle anderen Knoten sind entweder logische Gatter oder sog. Zwischenereignisse, welche durch Kombination mehrerer anderer Ereignisse entstehen. Die als Top-Level-Event bezeichnete Wurzel des Baumes ist ebenfalls ein solches Zwischenereignis und steht i. d. R. für den Systemausfall.

Beispiel

Seien X_1, X_2, und X_3 boolesche Variablen, welche den Ausfall der Komponenten eines 2-von-3-Systems beschreiben. Es ergibt sich somit für den Ausfall des Gesamtsystems (wobei die UND-Operatoren nicht ausgeschrieben sind – analog zum in der Mathematik üblicherweise weggelassenen Multiplikationspunkt):

$$X_{2v3} = X_1X_2 \vee X_1X_3 \vee X_2X_3$$

Als Fehlerbaum lässt sich diese boolesche Funktion nun wie in Abbildung 20 gezeigt darstellen:

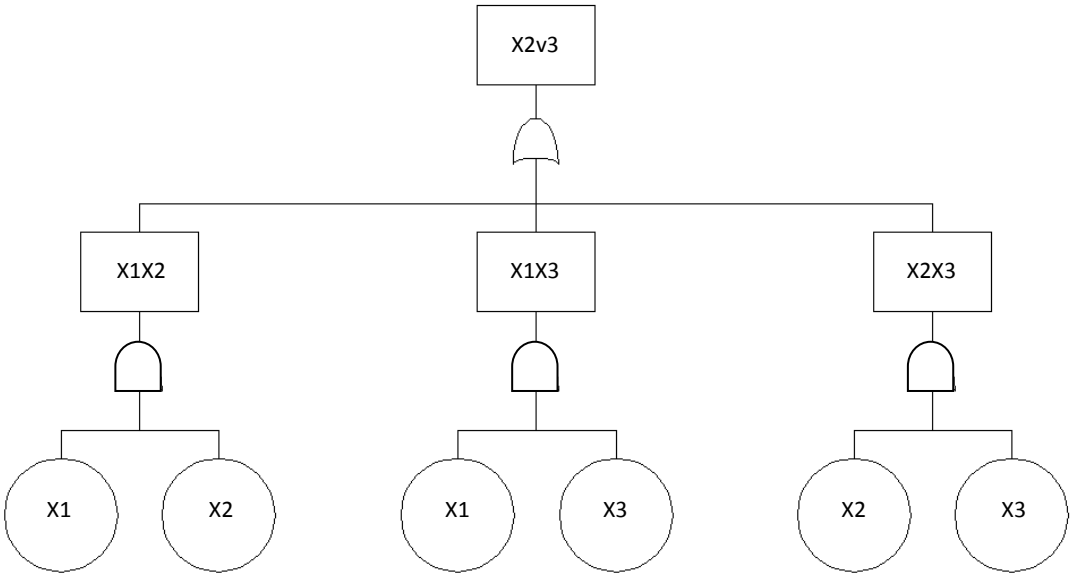


Abbildung 20: Fehlerbaum eines 2-von-3-Systems

Auch wenn in diesem einfachen Beispiel der Fehlerbaum umfangreicher ist, als die ihm zugrundeliegende Funktion, wird bereits hier offensichtlich, dass in größeren Systemen die grafische Darstellung eine bessere Übersicht bieten kann.

Die Tatsache, dass in Fehlerbäumen die Bedingungen für einen Ausfall (und nicht etwa für das einwandfreie Funktionieren des Systems) dargestellt sind, liegt darin begründet, dass Fehler im Allgemeinen leichter zu spezifizieren sind als ihre Abwesenheit (Haasl, Roberts, Vesely, & Goldberg, 1981), dass meist weniger Ursachen für einen Fehler als notwendige Randbedingungen für eine korrekte Funktion notwendig sind (ebd.) und dass nur bei Verwendung von typischerweise kleinen Fehlerwahrscheinlichkeiten ein in der Praxis unumgängliches Abschätzen der Ausfallwahrscheinlichkeit durch das Weglassen von Termen höherer Ordnung möglich ist (Walter & Schneeweiss, 2005).

Auswertung – Qualitative Analyse

Jede kleinstmögliche Menge an Primärereignissen, deren gleichzeitiger Eintritt zu einem Systemausfall führen würde, wird als **minimaler Schnitt** *(engl.: minimal cut set)* bezeichnet (Echtle, 2008). Ein Fehlerbaum hat durch seine Definition eine eindeutige endliche Menge an minimalen Schnitten.

Die Größe dieser minimalen Schnitte (d. h. die Mächtigkeit der sie bestimmenden Mengen) ist insofern von Bedeutung, als sie eine Folgerung darüber zulässt, durch wie viele simultane Ausfälle von Komponenten es zum Systemversagen kommt. So kann, wenn alle minimalen Schnitte aus mindestens zwei Ereignissen bestehen, gefolgert werden, dass kein Einzelfehler zum Systemversagen führt. Ferner lässt sich aus der Zusammensetzung der minimalen Schnitte erkennen, wie hoch die Verwundbarkeit des Systems bezüglich sog. Common Cause-Fehler (vgl. Kapitel 2) ist.

Fehlerbäume sind zunächst demnach rein qualitative Modelle. Werden die Komponentenausfälle mit Wahrscheinlichkeiten versehen, lassen sich, wie im folgenden Abschnitt dargestellt, jedoch auch quantitative Aussagen treffen.

Auswertung – Quantitative Analyse

Die Wahrscheinlichkeit eines Systemausfalls aufgrund eines bestimmten minimalen Schnitts ist das Produkt der (als stochastisch unabhängig angenommenen) Primärereignisse in diesem Schnitt. Sei nun $\bigcup_{i=1}^{n} M_i$ die Menge der minimalen Schnitte. Dann ist die Wahrscheinlichkeit des Systemausfalls $P = P(M_1 \cup M_2 \cup \ldots \cup M_n)$. Da sich die minimalen Schnitte jedoch nicht notwendigerweise gegenseitig ausschließen, können die Einzelwahr-

scheinlichkeiten nicht aufaddiert werden, sondern liefern nach der Formel von Pointcaré-Sylvester:

$$\begin{aligned} P(M_1 \cup M_2 \cup \ldots \cup M_n) = & \sum_{i=1}^{n} P(M_i) \\ & - \sum_{i=2}^{n} \sum_{j=1}^{i-1} P\left(M_i \cap M_j\right) \\ & + \sum_{i=3}^{n} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} P\left(M_i \cap M_j \cap M_k\right) \\ & - \cdots + (-1)^{n-1} P(M_1 \cap M_2 \cap \ldots \cap M_n) \end{aligned}$$

Zu beachten ist, dass sich auch die UND-Verknüpfung mehrerer minimaler Schnitte nicht aus dem Produkt der Einzelwahrscheinlichkeiten berechnet. Da die minimalen Schnitte Primärereignisse gemeinsam haben können, ist vielmehr das Produkt aller Primärereignisse dieser Schnitte heranzuziehen.

Selbst für einfache Systeme wird die Anzahl der so zu berechnenden Terme insgesamt sehr groß. Man nutzt üblicherweise die folgende Rechenvorschrift zur Annäherung:

$$\begin{aligned} P_1 &= \sum_{i=1}^{n} P(M_i) \\ P_2 &= P_1 - \sum_{i=2}^{n} \sum_{j=1}^{i-1} P\left(M_i \cap M_j\right) \\ P_3 &= P_2 + \sum_{i=3}^{n} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} P\left(M_i \cap M_j \cap M_k\right) \\ &\ldots \end{aligned}$$

Da die P_i jeweils abwechselnd immer genauer werdende obere und untere Schranken für P bilden (Formal Software Construction, 2005), kann eine Näherung für die gesuchte Gesamtausfallwahrscheinlichkeit mit vertretbarer Rechenzeit bestimmt werden. Aus den Wahrscheinlichkeiten der einzelnen minimalen Schnitte lässt sich sodann auch ihr relativer Anteil an der Ausfallwahrscheinlichkeit für das Gesamtsystem bestimmen.

Umrechnung von Raten in Wahrscheinlichkeiten

Oftmals sind für die Primärereignisse keine Wahrscheinlichkeiten, sondern konstante Ausfallraten λ (Ausfälle pro Zeiteinheit) gegeben, so dass sich eine exponentialverteilte Lebensdauer ergibt (vgl. Kapitel 2). Die zugehörige Fehlerwahrscheinlichkeit ist folglich:

$$F(t) = 1 - e^{-\lambda \cdot t}$$

Für einen beliebigen Zeitpunkt T lassen sich in diesem Fall durch Einsetzen in obige Formel die Fehlerwahrscheinlichkeiten $F(T)$ der einzelnen Primärereignisse ermitteln, woraus dann die Wahrscheinlichkeitsverteilung des Top-Level-Events nach dem oben dargestellten Verfahren bestimmt werden kann.

5.2.1.2 Modell

Durch das Aufstellen der jeweiligen Fehlerbäume können nun die in Kapitel 4.3.3 vorgestellten Systemvarianten der elektronisch geregelten Lenkung mit Dedizierter Redundanz (Abbildung 11) und Entfernter Redundanz (Abbildung 12) miteinander verglichen werden. Dieser Vergleich bezieht sich zum einen auf die jeweiligen minimalen Schnitte und die Ordnung der Schnitte, insbesondere auf die Frage nach eventuell unerkannten „Single Points of Failure“. Zum anderen ergibt sich die Möglichkeit, festzustellen, ob sich die Gesamtüberlebenswahrscheinlichkeit des Systems durch den Übergang von Dedizierter Redundanz zu Entfernter Redundanz ändert. Bevor das Ergebnis dieser Analyse vorgestellt wird, sollen jedoch zunächst beide Modelle beschrieben werden.

Der gewählte Detaillierungsgrad richtet sich nach dem in Tabelle 1 vorgestellten Fehlermodell und der Verfügbarkeit quantitativer Daten für die Wahrscheinlichkeit eines Ausfalls einzelner Komponenten. So werden Brückenschaltungen zur Ansteuerung der Motoren hier mit der zugehörigen ECU zusammengefasst; Kabel werden derjenigen Komponente, von der sie ausgehen, zugeordnet. Die Ausfallraten der Komponenten werden in beiden Modellen als identisch angenommen, auch wenn sich hierdurch eine gewisse Vergröberung ergibt (zusätzliche Fehlermöglichkeiten bei der Signaturerzeugung/-prüfung werden nicht berücksichtigt). Qualitative Aussagen bleiben jedoch hiervon ohnehin unberührt, quantitative Aussagen sind stets im Sinne von Größenordnungen zu verstehen.

Dedizierte Redundanz

Der Fehlerbaubaum für die Systemvariante mit Dedizierter Redundanz (hier als R_{dedic} bezeichnet) wird in seiner Gesamtstruktur in Abbildung 21 dargestellt. Der Baum gliedert sich unterhalb des Wurzelereignisses „Ausfall der Lenkfunktion“ in vier Teilbäume (mit römischen Ziffern gekennzeichnet). Die vier Teilbäume stehen für die nachfolgend genannten Situationen, welche jede für sich genommen zu einem Versagen der Lenkfunktion führen würden:

- Einer der beiden Motoren bewegt sich gegenläufig zur vorgegebenen Richtung, ohne passiviert zu werden (Teilbäume I und II): Am Differential ergibt sich als Summe der beiden gegenläufigen Bewegungen ein Stillstand der Ausgangswelle.
- Beide Motoren werden so stark gebremst, dass die Lenkung insgesamt zu träge reagiert (Teilbaum III, gegliedert in III.a und III.b): Die Sollvorgabe des Fahrers kann nicht rechtzeitig umgesetzt werden.
- Das Bussystem fällt aus (Teilbaum IV): Es ist keine Kommunikation mehr möglich, so dass die Sollvorgabe unberücksichtigt bleiben muss und der Regelkreis insgesamt unterbrochen ist.

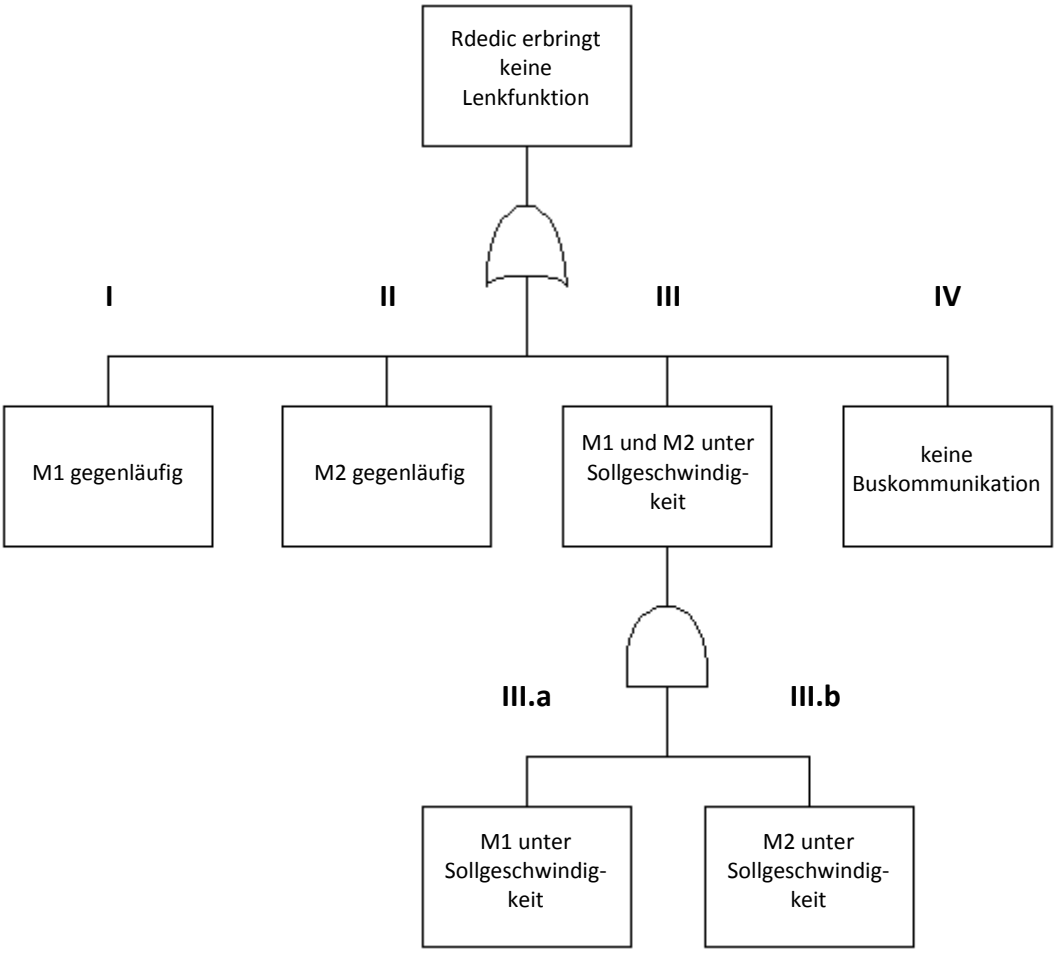


Abbildung 21: Fehlerbaum der Lenkung mit Dedizierter Redundanz

Betrachtet wird nun zunächst Teilbaum I (s. Abbildung 22). Das durch ihn beschriebene Ereignis „M_1 gegenläufig“ tritt genau dann ein, wenn Motor M_1 in die falsche Richtung angesteuert, aber nicht passiviert wird.

Eine falsche Ansteuerung ergibt sich entweder aus einem Fehler bei der Berechnung des Steuerbefehls in ECU_1 oder aus einer von Sensor P_1 ausgehenden falschen Positionsangabe, welche die Berechnung des korrekten Steuerbefehls durch ECU_1 unmöglich macht.

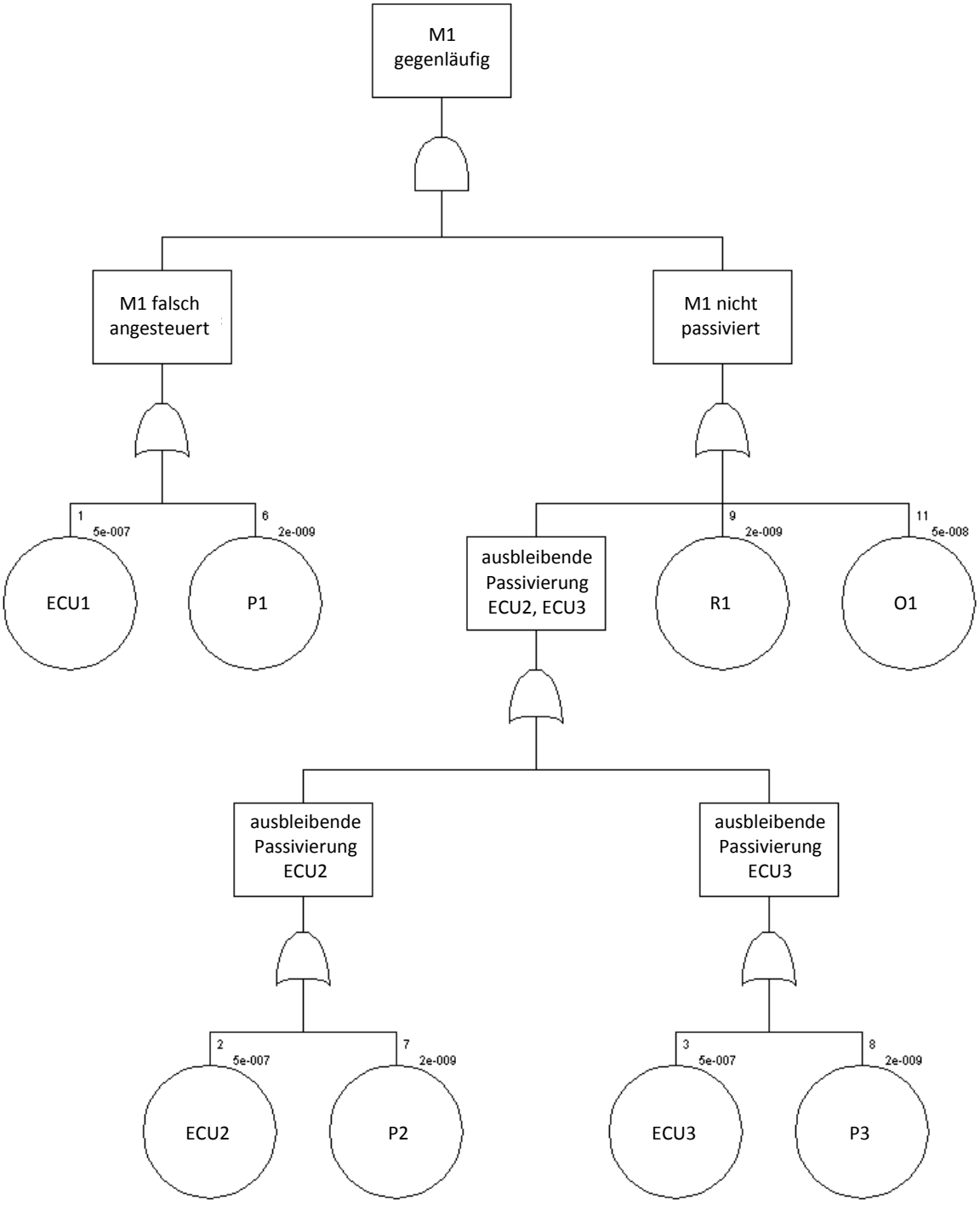


Abbildung 22: Teilbaum I, Dedizierte Redundanz

Eine ausbleibende Passivierung von Motor M_1 erklärt sich unmittelbar durch eine defekte zugehörige Endstufe O_1 oder durch einen Fehler des für die Passivierungsentscheidung herangezogenen Rotationssensors R_1 oder schließlich dadurch, dass sich entweder Knoten 2 oder Knoten 3 aufgrund eines Fehlers nicht für eine Passivierung entscheidet (M_1 wird nur passiviert, wenn beide Knoten zugleich für die Passivierung stimmen).

Von Knoten 2 geht keine Passivierung aus, wenn die zugehörige ECU_2 selbst fehlerhaft ist oder wenn sie von Sensor P_2 eine falsche Positionsangabe erhält und daher die Notwendigkeit einer Passivierung nicht erkennen kann. Für Knoten 3 gilt diese Situation analog: Hier kann ein Fehler in ECU_3 oder auch ein Fehler P_3 betreffend die Passivierung verhindern.

Für den in Teilbaum II (s. Abbildung 23) dargestellten Fall einer gegenläufig zur Sollvorgabe stattfindenden Ansteuerung von Motor M_2 gilt die oben beschriebene Situation analog: Auch dieser Motor kann sich nur dann gegenläufig zur vorgegebenen Richtung bewegen, wenn er falsch angesteuert und zugleich aufgrund eines weiteren Fehlers nicht passiviert wird.

Eine falsche Ansteuerung ergibt sich hier potentiell durch Fehler in ECU_2 oder P_2. Das Ausbleiben einer Passivierung kann entstehen durch einen Fehler der zugehörigen Endstufe O_2, des zur Passivierungsentscheidung herangezogenen Rotationssensors R_2 oder einer fehlerhaften Einschätzung der Situation durch die an der Passivierung beteiligten Knoten 1 und Knoten 3. Diese wiederum kann ihre Ursache haben in Fehlern von ECU_1 oder P_1 bzw. ECU_3 oder P_3.

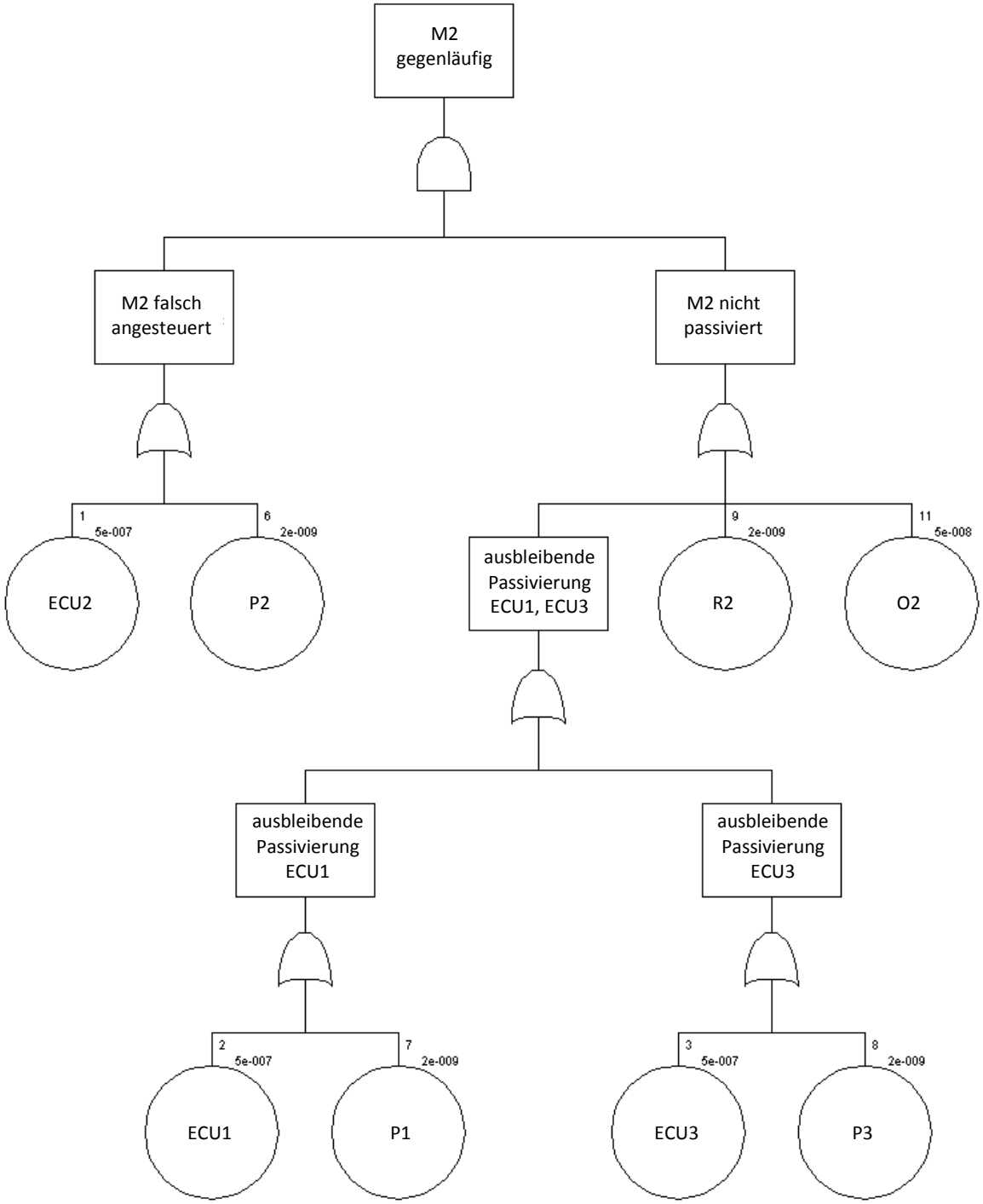


Abbildung 23: Teilbaum II, Dedizierte Redundanz

Teilbaum III beschreibt, wie oben angeführt, den Fall, dass beide Motoren die von ihnen ausgehenden Wellen nicht mit der geforderten Sollgeschwindigkeit in Bewegung versetzen. Zunächst wird nun in Teilbaum III.a (vgl. Abbildung 24) beschrieben, wie diese Situation für Motor M_1 eintreten kann.

Motor M_1 kann unter die für ein rechtzeitiges Reagieren der Lenkung erforderliche Sollgeschwindigkeit fallen, wenn er passiviert wird, er aufgrund einer falschen Ansteuerung inaktiv bleibt oder er selbst defekt (z. B. blockiert) ist.

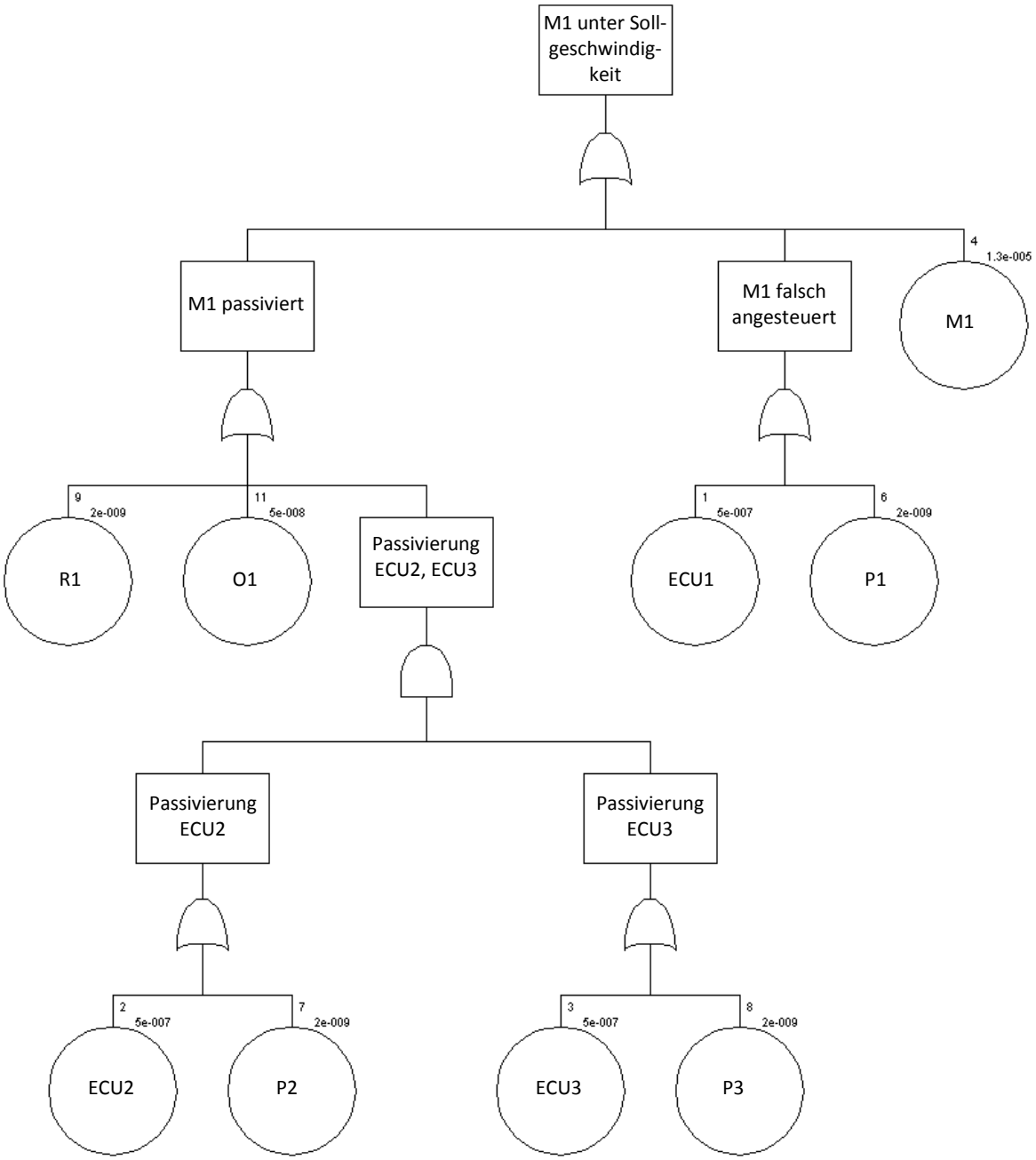


Abbildung 24: Teilbaum III.a, Dedizierte Redundanz

Eine Passivierung von Motor M_1 kann durch eine defekte Endstufe O_1, durch einen Fehler des zur Passivierungsentscheidung herangezogenen Rotationssensors R_1 oder durch gleichzeitige Passivierung ausgehend von Knoten 2 und Knoten 3 hervorgerufen sein (beide

Stimmen sind für eine Passivierung notwendig). Die Passivierung durch Knoten 2 und 3 ergibt sich ihrerseits durch Fehler in ECU_2 oder P_2 bzw. ECU_3 oder P_3, indem entweder ein falscher Steuerbefehl berechnet wird oder sich die betroffenen ECUs fehlerbedingt stets für eine Passivierung entscheiden.

Eine falsche Ansteuerung des Motors M_1 kann ebenfalls zu einem Nichterreichen der Sollgeschwindigkeit führen, beispielsweise wenn der Motor stets den Steuerbefehl „keine Positionsänderung“ erhält. Eine solche Situation kann entweder unmittelbar durch einen Fehler in ECU_1 verursacht werden oder dadurch, dass diese ECU von dem ihr zugeordneten Positionssensor P_1 fehlerhafte Werte erhält.

Die in Teilbaum III.b (vgl. Abbildung 25) dargestellte Situation für Motor M_2 verhält sich analog: Auch dieser Motor kann unter die vorgegebene Sollgeschwindigkeit fallen, wenn er passiviert oder falsch angesteuert wird oder selbst defekt (d. h. blockiert) ist.

Eine Passivierung von M_2 kann hervorgerufen werden durch Fehler in Rotationssensor R_2, Endstufe O_2 oder Knoten 1 und Knoten 3 zugleich (auch hier sind beide Stimmen für eine Passivierung erforderlich). Eine falsche Ansteuerung des Motors kann durch Fehler der entsprechenden ECU_2 (Steuerbefehl „keine Positionsänderung“) oder des ihr zugeordneten Positionssensors P_2 entstehen.

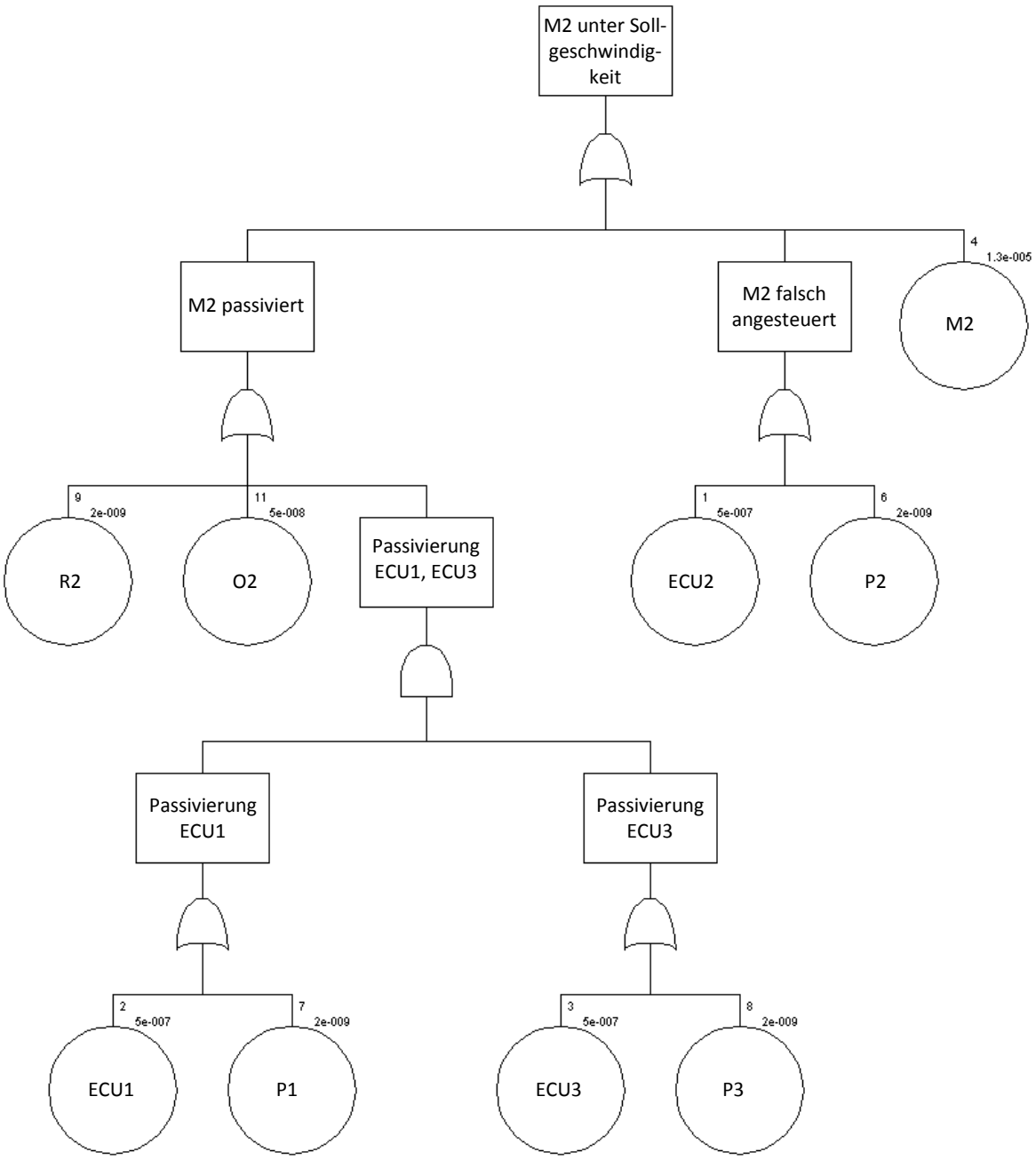


Abbildung 25: Teilbaum III.b, Dedizierte Redundanz

Als letzte Möglichkeit für ein Nichterbringen der Lenkfunktion in der Systemvariante mit Dedizierter Redundanz beschreibt Teilbaum IV (s. Abbildung 26) einen Komplettausfall des Bussystems. Ein solcher Ausfall wird hier vereinfachend durch den gleichzeitigen Ausfall beider Buskanäle B_1 und B_2 modelliert.

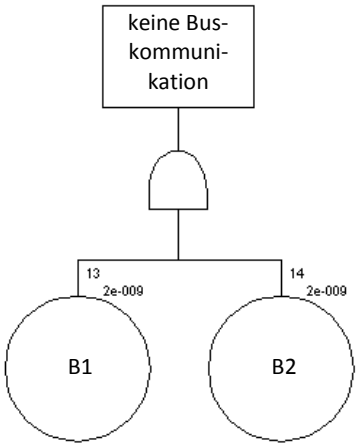


Abbildung 26: Teilbaum IV, Dedizierte Redundanz

Entfernte Redundanz

Die Systemvariante mit Entfernter Redundanz (hier R_{remote} genannt) unterscheidet sich im Wesentlichen dadurch, dass Positionssensor P_3 wegfällt und seine Funktion durch andere Komponenten ersetzt wird, deren Fehler nun anstelle eines Ausfalls von P_3 zu berücksichtigen sind. Des Weiteren ist, anders als bei Dedizierter Redundanz, zu beachten, dass eine fehlerhafte ECU nun sämtliche nachgelagerte Peripherie von der Kommunikation abschneiden kann, indem sie Nachrichten nicht an diese weiterleitet. Die Grundstruktur des daraus entstehenden Fehlerbaums ist in nachfolgender Abbildung 27 dargestellt. Auch hier ergibt sich das Wurzelereignis „Lenkfunktion nicht erbracht“ aus vier Teilbäumen, welche ebenfalls mit römischen Ziffern bezeichnet sind. Die Teilbäume unterhalb des Wurzelereignisses sind zunächst völlig identisch zur Systemvariante mit Dedizierter Redundanz: Das Nichterbringen der Lenkfunktion kann dadurch begründet sein, dass einer der beiden Motoren sich in die falsche Richtung bewegt ohne passiviert zu werden (Teilbäume I und II), dass beide Motoren unter die vorgegebene Sollgeschwindigkeit fallen (Teilbaum III, erneut aufgeteilt in die Teilbäume III.a und III.b) oder dass ein Komplettausfall des Bussystems zu verzeichnen ist und dieser jegliche Kommunikation verhindert (Teilbaum IV).

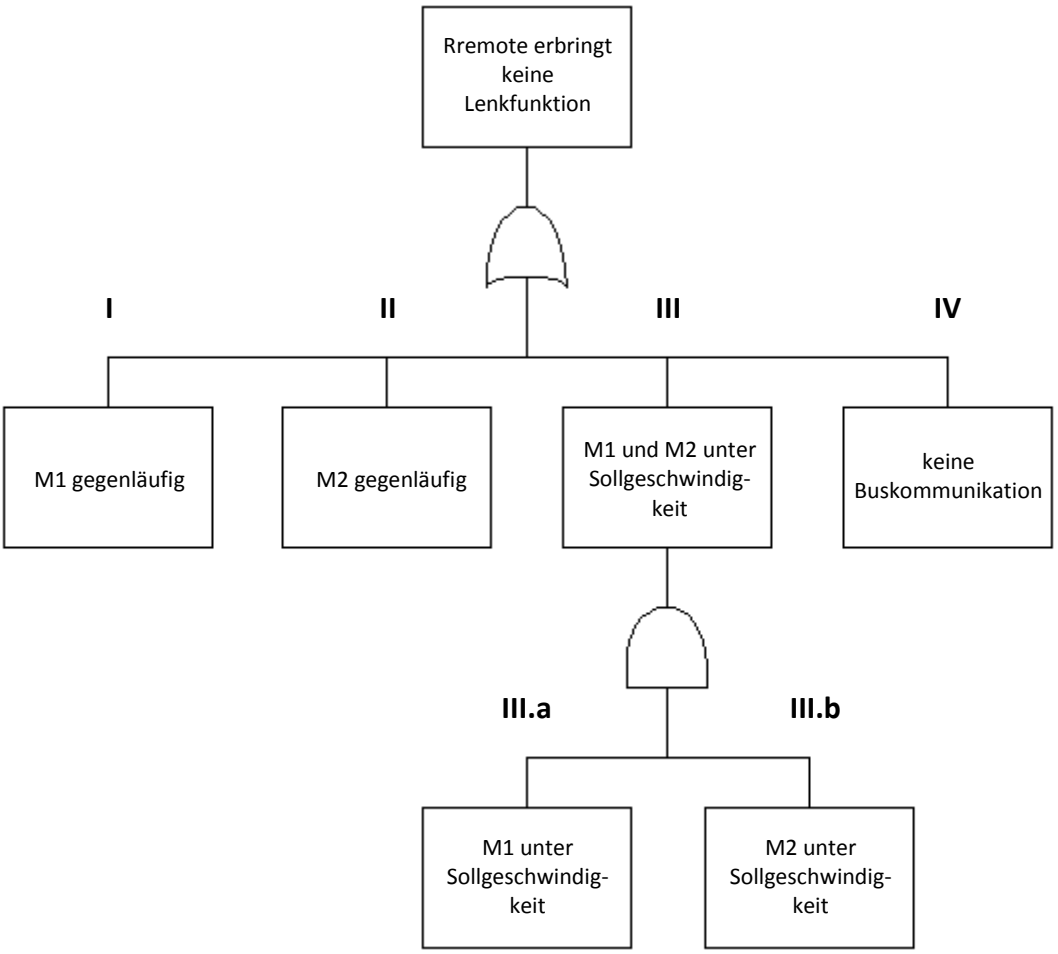


Abbildung 27: Fehlerbaum der Lenkung mit Entfernter Redundanz

Auch für dieses System wird nun zunächst Teilbaum I betrachtet (vgl. Abbildung 28). Motor M_1 bewegt sich abermals gegenläufig zur vorgegebenen Richtung, wenn er falsch angesteuert und nicht passiviert wird. Eine falsche Ansteuerung kann sich dadurch ergeben, dass der betreffende Positionssensor P_1 oder die ansteuernde ECU_1 (hier auch, indem sie die Kommunikation und damit den Regelkreis unterbricht) fehlerhaft sind.

Erneut kann eine Passivierung verhindert werden durch Fehler der entsprechenden Endstufe O_1 oder des für die Passivierungsentscheidung herangezogenen Rotationssensors R_1. Sind diese Komponenten fehlerfrei, muss, ebenso identisch zur Systemvariante mit Dedizierter Redundanz, entweder die Passivierung durch Knoten 2 oder die Passivierung durch Knoten 3 ausbleiben. Dies kann geschehen durch Fehler in ECU_2 oder P_2 bzw. ECU3 oder P_3.

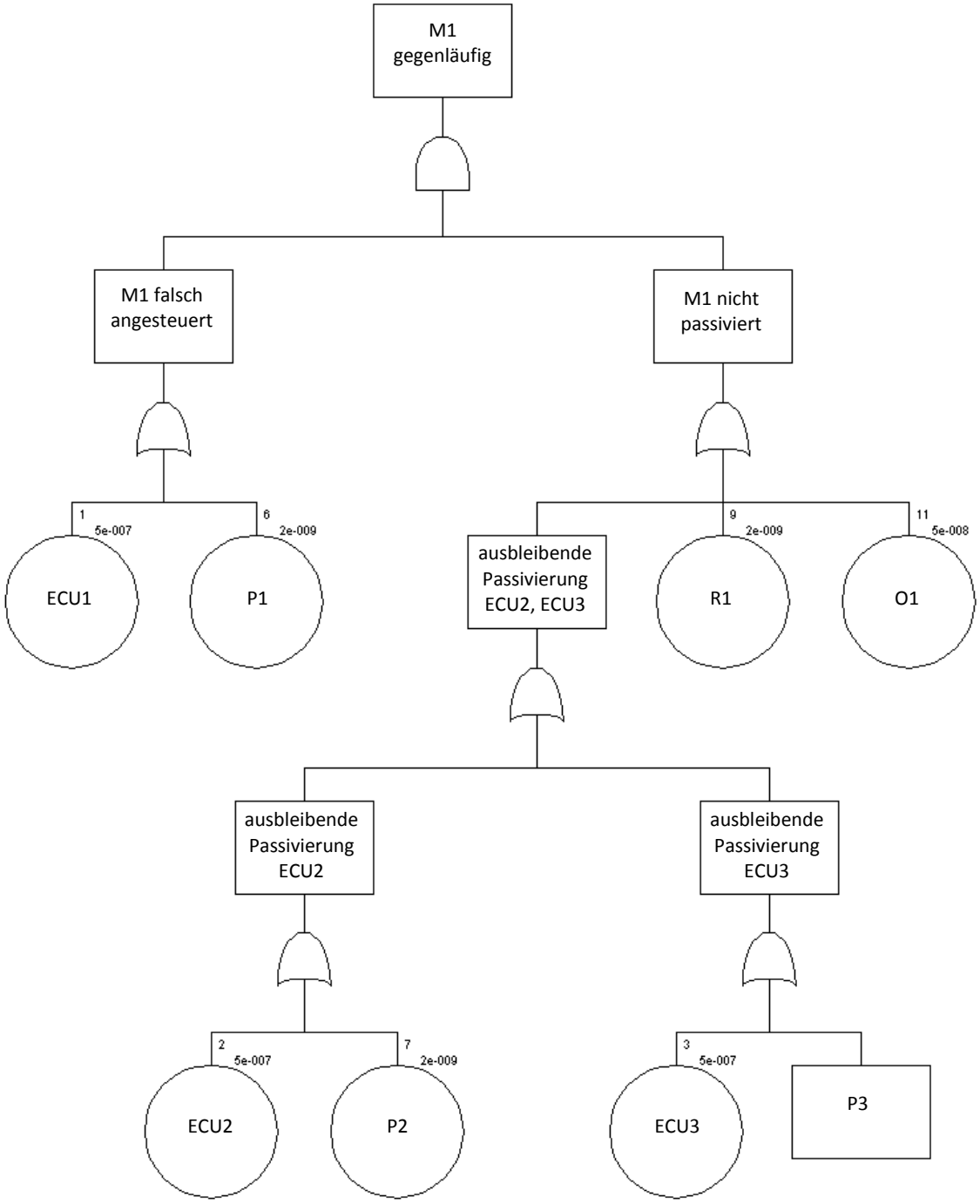


Abbildung 28: Teilbaum I, Entfernte Redundanz

Die Besonderheit in der Systemvariante mit Entfernter Redundanz liegt nun darin, dass Positionssensor P_3 nicht als Komponente vorliegt, sondern in seiner Funktion durch andere Komponenten nachgebildet wird, nämlich durch die Rotationssensoren R_1 und R_2. Dies kann etwa erfolgen, indem ECU_3 zunächst einen Näherungswert ermittelt, etwa durch Berechnung des folgenden Zeitintegrals:

$$\tilde{P}_3 = \int_0^t \frac{R_1(x) + R_2(x)}{2} dx$$

Anschließend wird als P_3 ausgehend von der Menge der weiterhin existierenden Sensoren $\{P_1, P_2\}$ derjenige Positionswert ausgewählt, welcher am geringsten vom Näherungswert $\tilde{P}_3$ abweicht:

$$P_3 = \begin{cases} P_1, & \left|\tilde{P}_3 - P_1\right| < \left|\tilde{P}_3 - P_2\right| \\ P_2, & \left|\tilde{P}_3 - P_1\right| \geq \left|\tilde{P}_3 - P_2\right| \end{cases}$$

Insgesamt lässt sich also eine unzulässige Passivierung durch Knoten 3 mittels der folgenden booleschen Funktion darstellen:

$$ECU_3 \vee R_1 \vee P_3$$

mit $P_3 = \tilde{P}_3(P_1 \vee P_2) \vee (P_1 P_2)$ und $\tilde{P}_3 = R_1 \vee R_2$.

Einsetzen ergibt demzufolge:

$$ECU_3 \vee R_1 \vee ((R_1 \vee R_2) \wedge (P_1 \vee P_2)) \vee (P_1 P_2)$$

bzw. gekürzt:

$$ECU_3 \vee R_1 \vee (R_2(P_1 \vee P_2)) \vee (P_1 P_2)$$

Da Fehler von ECU_3 bzw. R_1 bereits im betreffenden Teilbaum berücksichtigt sind, lässt sich der verbleibende Term, wie in Abbildung 29 dargestellt, gemäß des folgenden Zusammenhangs abbilden: Zu einer durch einen fehlerhaften Wert P_3 verursachten Passivierung von Motor M_1 kommt es, wenn R_2 und mindestens ein Positionssensor oder beide Positionssensoren fehlerhaft sind.

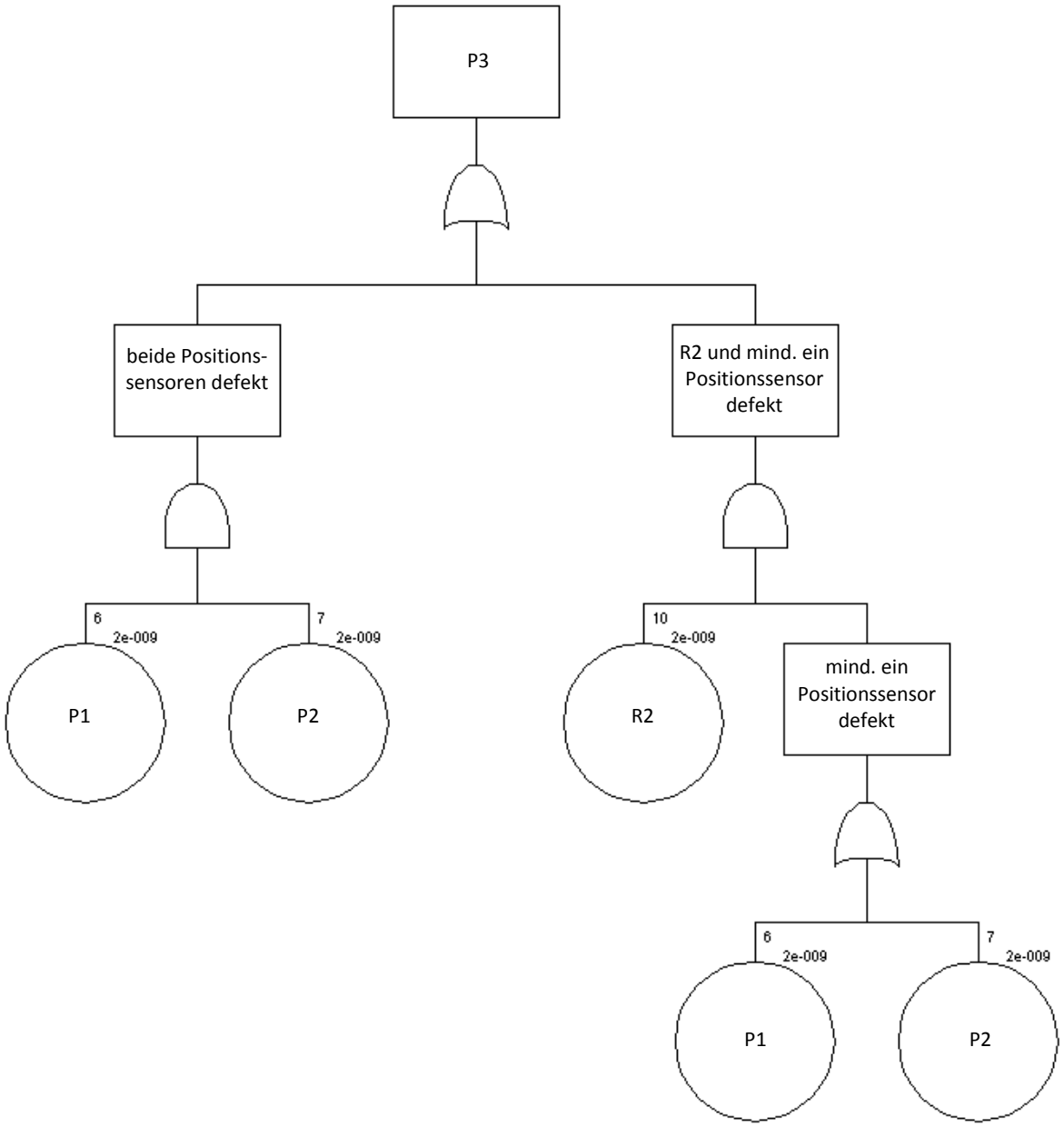


Abbildung 29: Berechnung von P3 zur Passivierung von M1

Die in Teilbaum II beschriebene Situation eines sich gegenläufig zur Sollvorgabe bewegenden Motors M_2 (s. Abbildung 30) verhält sich anlog zu Teilbaum I: Eine falsche Ansteuerung ergibt sich durch Fehler in ECU_2 oder P_2. Eine ausbleibende Passivierung kann entstehen durch Fehler in O_2, R_2 oder mindestens einem der beiden diesen Motor passivierenden Knoten. Die Berechnung von P_3 ergibt sich wie oben beschrieben, mit dem Unterschied, dass hier anstelle von R_1 der Rotationssensor R_2 bereits weiter oben im Fehlerbaum enthalten ist und daher bei den Fehlermöglichkeiten für P_3 nicht mehr berücksichtigt werden muss (ohne Abbildung).

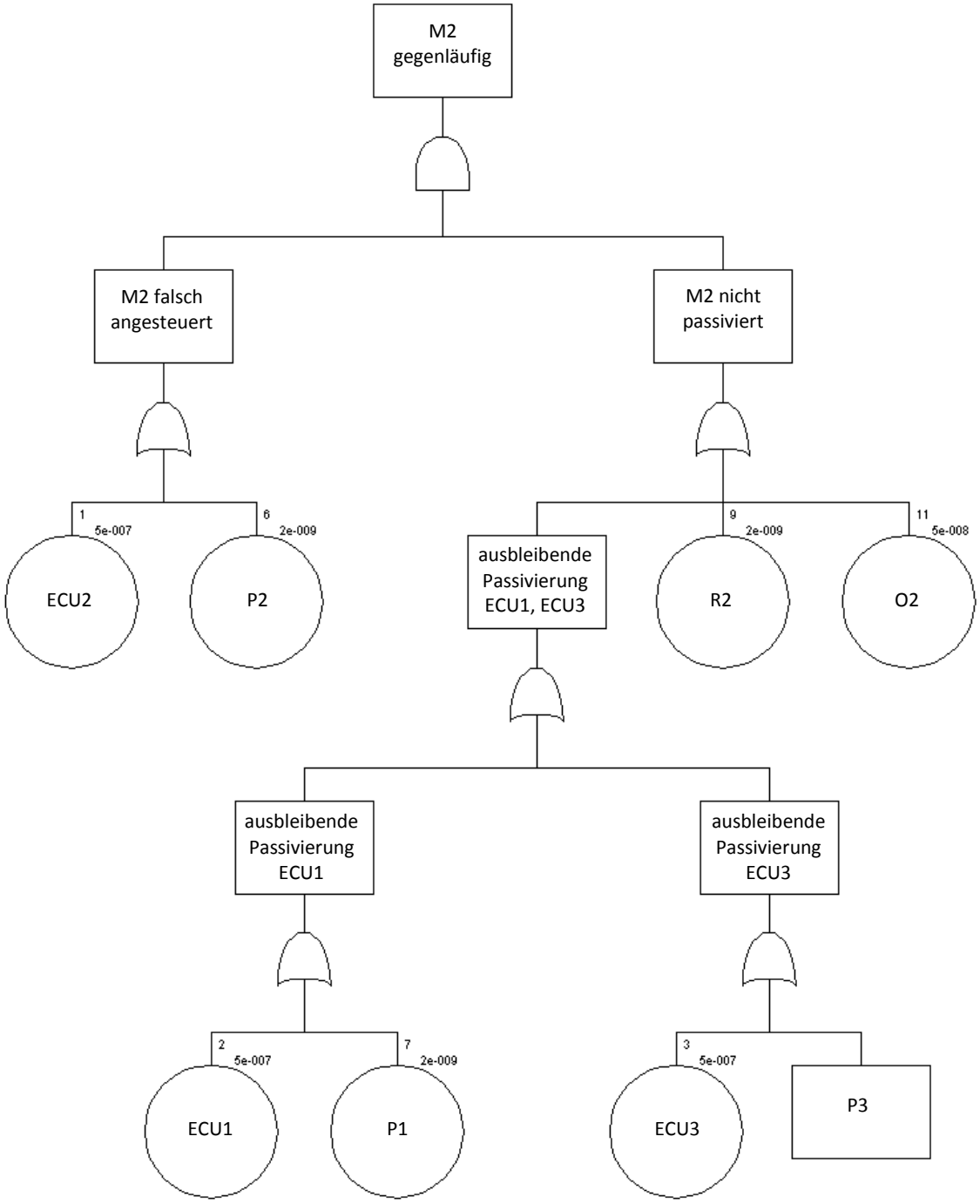


Abbildung 30: Teilbaum II, Entfernte Redundanz

Teilbaum III.a (vgl. Abbildung 31) beschreibt den Fall, dass Motor M_1 unter die Sollgeschwindigkeit fällt. Eine Möglichkeit hierfür ist erneut, dass Motor M_1 selbst fehlerhaft (also etwa blockiert) ist. M_1 kann aber auch durch Fehler in P_1 oder ECU_1 falsch angesteuert oder durch ECU_1 von der Kommunikation abgeschnitten werden. In letztgenannten Fall passiviert die zugehörige Endstufe O_1 Motor M_1 nach Ablauf eines Timeouts automatisch.

Die Passivierung kann zudem erfolgen durch einen Defekt in Endstufe O_1 selbst, durch einen Fehler in R_1, welcher den zur Passivierungsentscheidung herangezogenen Rotationswert liefert, oder durch gemeinsame Passivierung ausgehend von Knoten 2 und Knoten 3. Ursachen für eine Passivierung durch Knoten 2 sind Fehler in ECU_2 oder P_2. Knoten 3 entscheidet sich möglicherweise unberechtigt für eine Passivierung, wenn ECU_3 oder P_3 (auch hier nachgebildet durch die oben beschriebene Berechnung anhand der Werte anderer Sensoren) fehlerhaft sind.

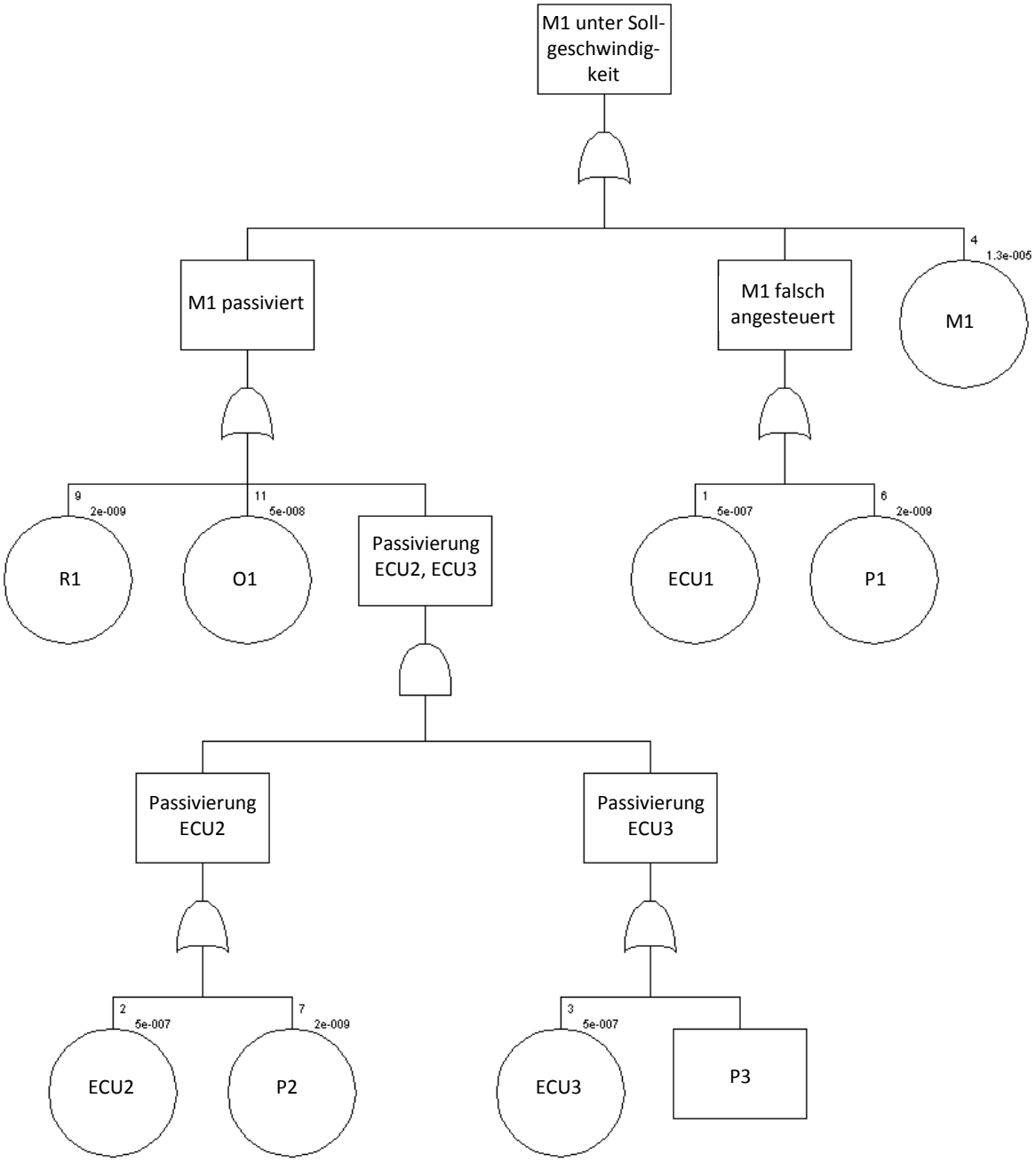


Abbildung 31: Teilbaum III.a, Entfernte Redundanz

Die in Teilbaum III.b (s. Abbildung 32) dargestellte Situation für Motor M_2 verhält sich spiegelbildlich: Dieser kann unter die vorgegebene Sollgeschwindigkeit fallen, wenn er selbst defekt ist, durch Fehler in ECU_2 oder P_2 falsch angesteuert oder von der Kommunikation abgeschnitten wird oder im Falle einer Passivierung durch Fehler in Endstufe O_2, Rotationssensor R_2 oder zugleich in Knoten 1 und Knoten 3.

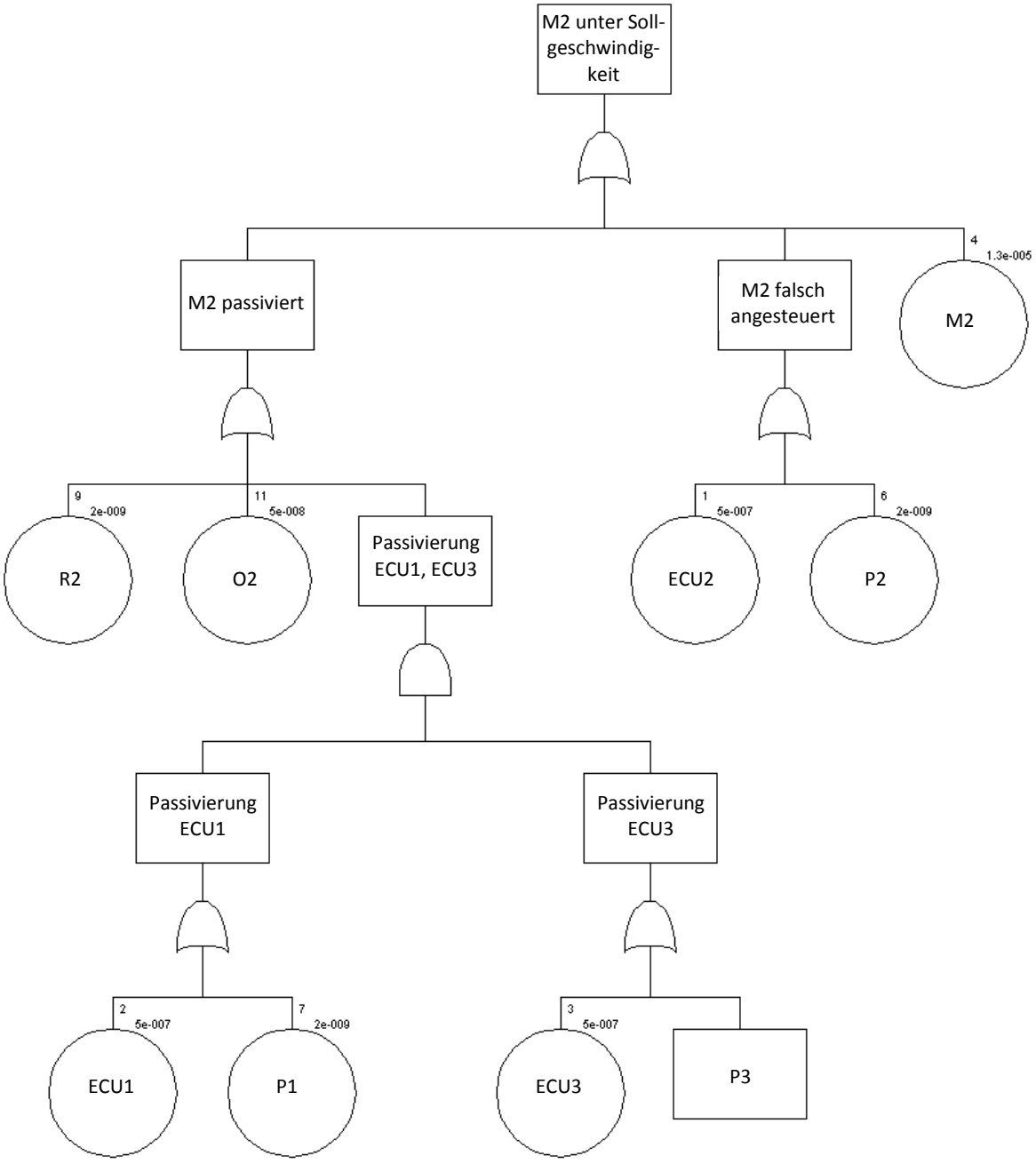


Abbildung 32: Teilbaum III.b, Entfernte Redundanz

Schließlich ist auch für die Systemvariante mit Entfernter Redundanz der in Teilbaum IV (vgl. Abbildung 33) dargestellte Komplettausfall des Bussystems zu berücksichtigen, welcher auch hier durch den gleichzeitigen Ausfall beider Buskanäle modelliert wird.

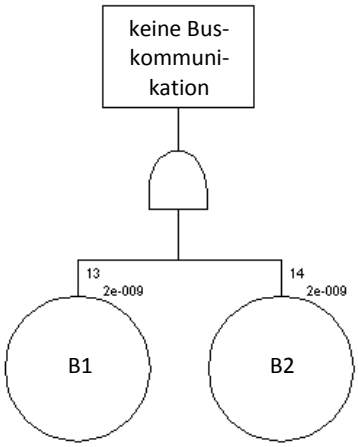


Abbildung 33: Teilbaum IV, Entfernte Redundanz

Beide Fehlerbäume sind sowohl als Bild wie auch als editierbare Datei im elektronischen Anhang der Arbeit enthalten.

5.2.1.3 Ergebnis

Die im vorangegangenen Kapitel beschriebenen Fehlerbäume beider Systemvarianten wurden mittels des freien Werkzeugs OpenFTA[9] der Firma Auvation zunächst auf ihre minimalen Schnitte hin untersucht. Im Ergebnis zeigt sich folgendes Bild:

Minimale Schnitte der Variante mit Dedizierter Redundanz

$\{B_1, B_2\}$, $\{ECU_1, ECU_2\}$, $\{ECU_1, ECU_3\}$, $\{ECU_1, M_2\}$, $\{ECU_1, O_1\}$, $\{ECU_1, O_2\}$, $\{ECU_1, P_2\}$, $\{ECU_1, P_3\}$, $\{ECU_1, R_1\}$, $\{ECU_1, R_2\}$, $\{ECU_2, ECU_3\}$, $\{ECU_2, M_1\}$, $\{ECU_2, O_1\}$, $\{ECU_2, O_2\}$, $\{ECU_2, P_1\}$, $\{ECU_2, P_3\}$, $\{ECU_2, R_1\}$, $\{ECU_2, R_2\}$, $\{ECU_3, P_1\}$, $\{ECU_3, P_2\}$, $\{M_1, M_2\}$, $\{M_1, O_2\}$, $\{M_1, P_2\}$, $\{M_1, R_2\}$, $\{M_2, O_1\}$, $\{M_2, P_1\}$, $\{M_2, R_1\}$, $\{O_1, O_2\}$, $\{O_1, P_1\}$, $\{O_1, P_2\}$, $\{O_1, R_2\}$, $\{O_2, P_1\}$, $\{O_2, P_2\}$, $\{O_2, R_1\}$, $\{P_1, P_2\}$, $\{P_1, P_3\}$, $\{P_1, R_1\}$, $\{P_1, R_2\}$, $\{P_2, P_3\}$, $\{P_2, R_1\}$, $\{P_2, R_2\}$, $\{R_1, R_2\}$

[9] http://www.openfta.com

Insgesamt gibt es somit für dieses System:

- 42 Schnitte zweiter Ordnung,
- keine Schnitte erster Ordnung, also keinen „Single Point of Failure“ und
- keine Schnitte höherer Ordnung.

Minimale Schnitte der Variante mit Entfernter Redundanz

$\{B_1, B_2\}$, $\{ECU_1, ECU_2\}$, $\{ECU_1, ECU_3\}$, $\{ECU_1, M_2\}$, $\{ECU_1, O_1\}$, $\{ECU_1, O_2\}$, $\{ECU_1, P_2\}$, $\{ECU_1, R_1\}$, $\{ECU_1, R_2\}$, $\{ECU_2, ECU_3\}$, $\{ECU_2, M_1\}$, $\{ECU_2, O_1\}$, $\{ECU_2, O_2\}$, $\{ECU_2, P_1\}$, $\{ECU_2, R_1\}$, $\{ECU_2, R_2\}$, $\{ECU_3, P_1\}$, $\{ECU_3, P_2\}$, $\{M_1, M_2\}$, $\{M_1, O_2\}$ $\{M_1, P_2\}$, $\{M_1, R_2\}$, $\{M_2, O_1\}$, $\{M_2, P_1\}$, $\{M_2, R_1\}$, $\{O_1, O_2\}$, $\{O_1, P_1\}$, $\{O_1, P_2\}$, $\{O_1, R_2\}$, $\{O_2, P_1\}$, $\{O_2, P_2\}$, $\{O_2, R_1\}$, $\{P_1, P_2\}$, $\{P_1, R_1\}$, $\{P_1, R_2\}$, $\{P_2, R_1\}$, $\{P_2, R_2\}$, $\{R_1, R_2\}$

Insgesamt gibt es hier:

- 38 Schnitte zweiter Ordnung,
- keine Schnitte erster Ordnung, d. h. keinen „Single Point of Failure“ und
- keine Schnitte höherer Ordnung.

Ferner bilden diese Schnitte eine echte Untermenge der minimalen Schnitte von R_{dedic}. Es gilt:

- Schnitte(R_{remote}) $\subset$ Schnitte(R_{dedic}) und
- Schnitte(R_{dedic}) = Schnitte(R_{remote}) $\cup$ Differenzschnitte,

wobei die Menge „Differenzschnitte“ aus den folgenden vier Schnitten besteht: $\{ECU_1, P_3\}$, $\{ECU_2, P_3\}$, $\{P_1, P_3\}$, $\{P_2, P_3\}$.

Aus der Teilmengen-Beziehung kann gefolgert werden, dass bei einem Ausfall des Systems mit Entfernter Redundanz auch ein entsprechendes System mit Dedizierter Redundanz ausfallen würde. Umgekehrt gibt es aber Fälle, in denen das System mit Dedizierter Redundanz ausfällt, aber das entsprechende System mit Entfernter Redundanz noch korrekt arbeitet. Daraus folgt: Das System mit Entfernter Redundanz weist stets eine höhere Zuverlässigkeit auf als ein entsprechendes System mit Dedizierter Redundanz. Dies gilt völlig unabhängig von den angenommenen Fehlerraten, jedoch ist vorauszusetzen, dass einander entsprechende Komponenten in beiden Systemen gleich zuverlässig sind. Insbesondere darf

ein Sensor bzw. Aktuator, der mit Signaturen arbeitet, nicht unzuverlässiger sein als ein Gerät, welches keine Signaturen verwendet.

Aus dieser Aussage geht nicht hervor, um welchen Betrag die Zuverlässigkeit verbessert worden ist. Es lässt sich aber ein wichtiges Argument ableiten, das für die Verwendung von Entfernter Redundanz spricht: Sie ist nicht schlechter als Dedizierte Redundanz, verursacht aber einen geringeren Redundanzaufwand.

Um die Zuverlässigkeit zu quantifizieren, wurden nun zunächst gemäß (U.S. Department of Defense, 1991) die in Tabelle 2 aufgeführten Ausfallraten für die einzelnen Komponenten herangezogen:

Tabelle 2: Angenommene Ausfallraten der Komponenten

Komponente	**Ausfallrate**
ECU	$5{,}00 \cdot 10^{-7}$ / h
Motor	$1{,}30 \cdot 10^{-5}$ / h
Bus	$2{,}00 \cdot 10^{-9}$ / h
Endstufe	$5{,}00 \cdot 10^{-8}$ / h
Sensor	$2{,}00 \cdot 10^{-9}$ / h

Für eine Berechnung der Überlebenswahrscheinlichkeit des Gesamtsystems wurden anhand dieser Ausfallraten sodann wie in Abschnitt 5.2.1.1 beschrieben die in Tabelle 3 enthaltenen Fehlerwahrscheinlichkeiten der Komponenten $F(t)$ zu ausgewählten Zeitpunkten $t \in \{100, 500, 1.000, 5.000, 10.000, 50.000, 100.000, 150.000, 200.000\}$berechnet:

Tabelle 3: Fehlerwahrscheinlichkeiten der Komponenten

	F(100)	**F(500)**	**F(1000)**	**F(5000)**	**F(10000)**	**F(50000)**	**F(100000)**	**F(150000)**	**F(200000)**
ECU	5,00E-05	2,50E-04	5,00E-04	2,50E-03	4,99E-03	2,47E-02	4,88E-02	7,23E-02	9,52E-02
Motor	1,30E-03	6,48E-03	1,29E-02	6,29E-02	1,22E-01	4,78E-01	7,27E-01	8,58E-01	9,26E-01
Sensor	2,00E-07	1,00E-06	2,00E-06	1,00E-05	2,00E-05	1,00E-04	2,00E-04	3,00E-04	4,00E-04
Endstufe	5,00E-06	2,50E-05	5,00E-05	2,50E-04	5,00E-04	2,50E-03	4,99E-03	7,47E-03	9,95E-03
Bus	2,00E-07	1,00E-06	2,00E-06	1,00E-05	2,00E-05	1,00E-04	2,00E-04	3,00E-04	4,00E-04

Hieraus wurden mittels des Werkzeugs OpenFTA für beide Systemvarianten die Fehlerwahrscheinlichkeiten des Gesamtsystems zum entsprechenden Zeitpunkt ermittelt, woraus sich unmittelbar auch die Überlebenswahrscheinlichkeiten $R(t) = 1 - F(t)$ ergeben. Hierfür wurde das weiter oben beschriebene Näherungsverfahren verwendet, wobei Schnittmengen von minimalen Schnitten bis zu einer Größe von sechs Termen berücksichtigt wurden. Die Ergebnisse sind in Tabelle 4 aufgeführt.

Tabelle 4: Fehler- und Überlebenswahrscheinlichkeiten des Gesamtsystems

	t=100	t=500	t=1000	t=5000	t=10000	t=50000	t=100000	t=150000	t=200000
F(t), Rded	1,84E-06	4,58E-05	1,84E-04	4,31E-03	1,60E-02	2,43E-01	5,52E-01	7,56E-01	8,73E-01
R(t), Rded	1,00E+00	1,00E+00	1,00E+00	9,96E-01	9,84E-01	7,57E-01	4,48E-01	2,44E-01	1,27E-01
F(t), Rrem	1,84E-06	4,58E-05	1,84E-04	4,55E-03	1,61E-02	2,43E-01	5,52E-01	7,57E-01	8,72E-01
R(t), Rrem	1,00E+00	1,00E+00	1,00E+00	9,95E-01	9,84E-01	7,57E-01	4,48E-01	2,43E-01	1,28E-01

Nunmehr können diese Werte einander gegenübergestellt werden, was in nachfolgender Abbildung 34 geschehen ist:

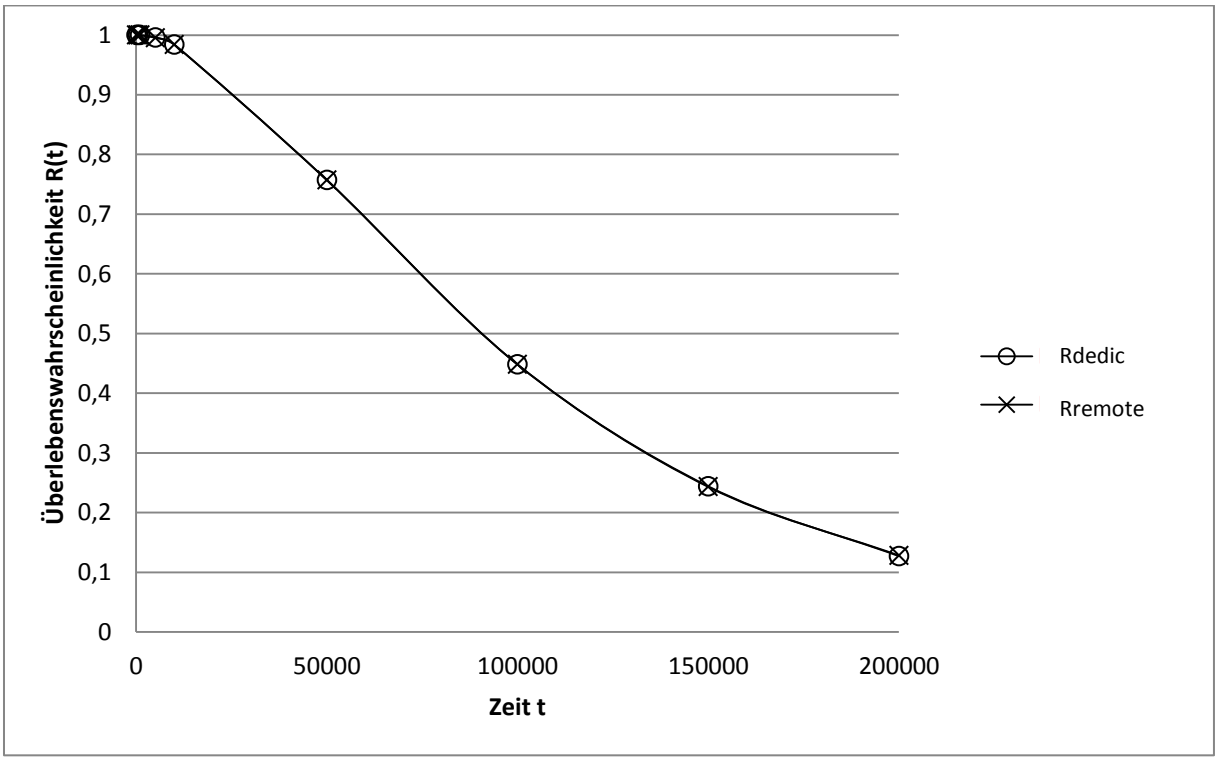


Abbildung 34: Graphen der Überlebenswahrscheinlichkeit beider Systeme

Die Berechnung zeigt, dass die Überlebenswahrscheinlichkeiten von Dedizierter und Entfernter Redundanz für alle Zeitpunkte in derselben Größenordnung liegen. Ferner zeigen die im elektronischen Anhang der Arbeit beinhalteten Ausgabedateien des Analysetools, dass die existierenden Abweichungen im Bereich der durch das näherungsweise Berechnungsverfahren gegebenen Ungenauigkeit liegen, so dass im Ergebnis von identischen Resultaten für beide Systemvarianten gesprochen werden kann.

Unabhängige Analysen mit gleichem Ergebnis wurden von Max Walter und Michael Pock (TU-München) sowie Sébastian Jacket und Olaf Malassé (Ecole Nationale Superieure d'Arts et Metiers, Metz) durchgeführt und auf der AR2TS ’09-Konferenz in Loughborough, UK, vorgestellt sowie im zugehörigen Tagungsband als (Echtle, Kimmeskamp, Jacquet, Malassé, Pock, & Walter, 2009) veröffentlicht.

Weitere Plausibilitätstests und Untersuchungen fanden während der vom Autor betreuten Projektseminararbeit von Marcel Imig (Imig, 2009) statt. In diesem Rahmen wurde u. a. gezeigt, dass sich am beschriebenen Ergebnis auch dann nichts ändert, wenn die anfälligsten Komponenten (die Motoren) als dem Perfektionskern zugehörig angenommen werden und andere Komponenten, wie etwa Sensoren, somit einen größeren Einfluss haben.

Insgesamt kann somit für das betrachtete Fallbeispiel und alle nach demselben Muster konstruierten Systeme Entfernte Redundanz als Dedizierter Redundanz mit Bezug auf die sich aus der Systemstruktur ergebenden Fehlertoleranzeigenschaften prinzipiell gleichrangig bewertet werden.

5.2.2 Formale Verifikation

5.2.2.1 Methode

Nachdem mit den im vorangegangenen Kapitel beschriebenen Untersuchungen die sich durch Verwendung Entfernter Redundanz ergebende Systemstruktur hinsichtlich ihrer Fehlertoleranzeigenschaften im Vergleich zum bisherigen Stand der Technik bewertet werden konnte, ist es Gegenstand des folgenden Kapitels, nun auch die tatsächliche Funktion des Beispielsystems der elektronisch geregelten Lenkung erstmals in vereinfachter Form zu modellieren und zu verifizieren. Für diesen Arbeitsschritt wurde das Werkzeug UPPAAL[10] verwendet. An seinem Beispiel sollen nun zunächst die Modellwelt und die Vorgehensweise erläutert werden.

[10] http://www.uppaal.org

Zu untersuchende Systeme werden in UPPAAL als Netzwerke synchron gekoppelter, zeitbehafteter, endlicher Automaten modelliert. Ein solcher Automat ist nach (Behrmann, David, & Larsen, 2004) definiert als Tupel $\mathcal{A} = (L, l_o, C, A, E, I)$ wobei L eine Menge von Stellen ist, $l_o \in L$ der Startzustand, C eine Menge von Uhren, A eine Menge von Aktionen, $E \subseteq L \times A \times B(C) \times 2^C \times L$ eine Menge von Transitionen zwischen jeweils zwei Stellen mit einer Aktion, einer Schaltbedingung sowie einer Menge dabei zurückzusetzender Uhren und $I: L \to B(C)$ eine Zuweisung von Invarianten zu den Stellen.

Hierbei ist $B(C)$ eine Menge von Konjunktionen über Ausdrücke der Form $x \bowtie c$ oder $x - y \bowtie c$ mit $x, y \in C$, $c \in \mathbb{N}$ und $\bowtie \in \{<, \leq, =, \geq, >\}$. Über der Menge der Aktionen A wird gemäß (Larsen, Petterson, & Yi, 1995) eine (partielle) Synchronisationsfunktion $f: (A \cup \{0\}) \times (A \cup \{0\}) \hookrightarrow A$ mit 0 als Symbol für „keine Aktion“ definiert, welche den gemeinsamen Zustandswechel mehrerer Automaten beschreibt.

UPPAAL erlaubt es zusätzlich, globale und lokale Variablen zu deklarieren und ihren Wert beim Schalten einer Transition in Form von sog. „Updates“ zu verändern. Auch die Deklaration von Datentypen und Funktionen ist möglich. Zudem wird das Konzept der Stellen insofern erweitert, als beim Schalten innerhalb von Stellen, welche als „urgent“ ausgezeichnet werden, keine Zeit verstreichen darf. Das Schlüsselwort „committed“ erzwingt ferner ein unmittelbares Verlassen der Stelle, bevor eine andere Transition schaltet. Zur syntaktischen Vereinfachung lassen sich Automaten – welche in UPPAAL auch als Prozesse bezeichnet werden – bei ihrer Erzeugung parametrisieren.

Zur Verifikation der erstellten Modelle mittels Zustandsraumexploration dient in UPPAAL eine Untermenge von CTL[11]. Die Ausdrücke dieser Abfragesprache bestehen aus Pfadquantoren, Temporaloperatoren und Aussagen über den Zustand des Systems (Variablenbelegungen, Constraints über Uhren, erreichte Stellen sowie das Schlüsselwort „deadlock“). Die an (David, Amnel, Stigge, & Ekberg, 2009) angelehnte, nachfolgende Tabelle 5 enthält eine Übersicht über die hierdurch möglichen Ausdrücke und ihre Bedeutung.

[11] Computation Tree Logic

Tabelle 5: Zulässige Ausdrücke der UPPAAL-Abfragesprache

Ausdruck	Beschreibung	Syntax UPPAAL
$\exists\Diamond\ \varphi$	es gibt einen Pfad, auf dem irgendwann φ gilt	`E<> p`
$\forall\Box\ \varphi$	auf allen Pfaden gilt immer φ	`A[] p`
$\exists\Box\ \varphi$	es gibt einen Pfad, auf dem immer φ gilt	`E[] p`
$\forall\Diamond\ \varphi$	auf allen Pfaden gilt irgendwann φ	`A<> p`
$\psi \rightarrow \varphi$	wann immer ψ gilt, wird schließlich φ gelten	`p-->q`

Als Beispiel für die Modellierung und Verifikation mit UPPAAL sei ein System aus (David, Amnel, Stigge, & Ekberg, 2009) aufgeführt, welches den Schutz eines kritischen Abschnitts (critical section, `CS`) nachbildet (s. Abbildung 35): Beide Automaten signalisieren beim Übergang von „`idle`“ zu „`want`“, dass sie eine Ressource anfordern (`req1:=1` bzw. `req2:=1`). Anschließend wird stets dem jeweils anderen Prozess der Vorrang gelassen (`turn:=2` bzw. `turn:=1`). Der kritische Abschnitt wird solange nicht betreten, wie der Prozess nicht an der Reihe ist (`turn==1` oder `turn==2`) und der andere Prozess den Abschnitt noch nicht verlassen hat (`req1==0` bzw. `req2==0`). Mit Verlassen des kritischen Abschnitts wird die Anforderung zurückgenommen: `req1:=0` und `req2:=0`.

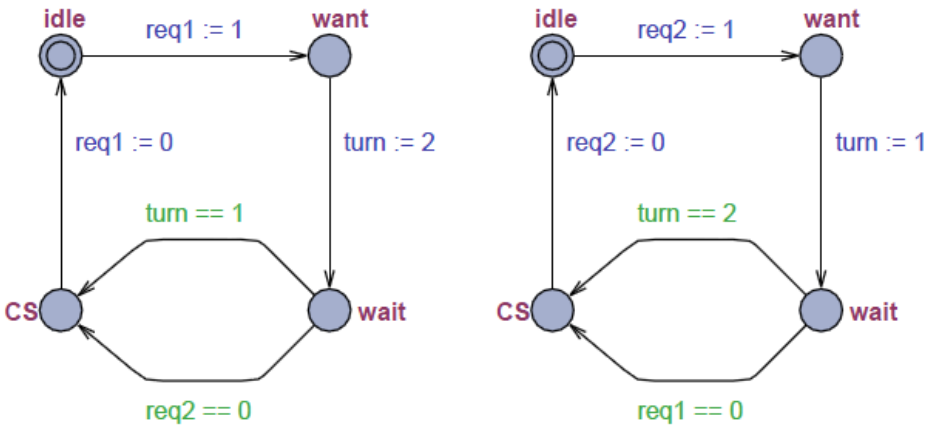


Abbildung 35: Gegenseitiger Ausschluss als UPPAAL-Modell

Das vorliegende Modell kann nun durch Simulation mit Hilfe des Tools „augenscheinvalidiert“ werden. UPPAAL ermöglicht jedoch auch eine formale Verifikation: So lässt sich durch Auswertung von Abfragen wie „`A[] not (P1.CS and P2.CS)`“ sicherstellen, dass der kritische Abschnitt nicht von beiden Prozessen zugleich betreten wird. Über die Ab-

frage „`E<> P1.CS`“ ist es etwa möglich, zu prüfen, ob der kritische Abschnitt von Prozess P1 grundsätzlich erreicht werden kann (dies ist ebenso für P2 durchzuführen).

Insgesamt bietet die mit UPPAAL gewählte (bzw. die von UPPAAL angebotene) Abstraktionsebene die Möglichkeit, das für Entfernte Redundanz benötigte algorithmische Verhalten für einen konkreten Fehlertoleranzgrad und an einem konkreten Beispiel formal zu verifizieren. Hierbei muss zwar von manchen Details abstrahiert werden, um den Zustandsraum handhabbar zu halten. Im Gegenzug dafür bietet sich jedoch die Möglichkeit, den gesamten Zustandsraum vollständig zu untersuchen und somit möglicherweise nicht beachtete Randfälle zu finden. Die in diesem Rahmen aus technischen Gründen nicht oder nur in vereinfachter Form behandelten Aspekte werden im Detail Gegenstand späterer Abschnitte sein.

Bevor im folgenden Kapitel das mit UPPAAL erstellte Modell des Beispielsystems der elektronisch geregelten Lenkung mit Entfernter Redundanz beschrieben wird, sei abschließend noch für eine detailliertere Darstellung der Möglichkeiten von UPPAAL auf (Larsen, Petterson, & Yi, 1997) verwiesen.

5.2.2.2 Modell

Das im Folgenden beschriebene UPPAAL-Modell der Lenkung mit Entfernter Redundanz basiert auf Unterrichtsmaterial zur Vorlesung „Modelle fehlertoleranter Systeme“ (Echtle, 2003) und wurde in der vorliegenden Form erstmals vorgestellt in (Echtle & Kimmeskamp, 2010). Einen ersten Überblick über das Modell liefert Abbildung 36:

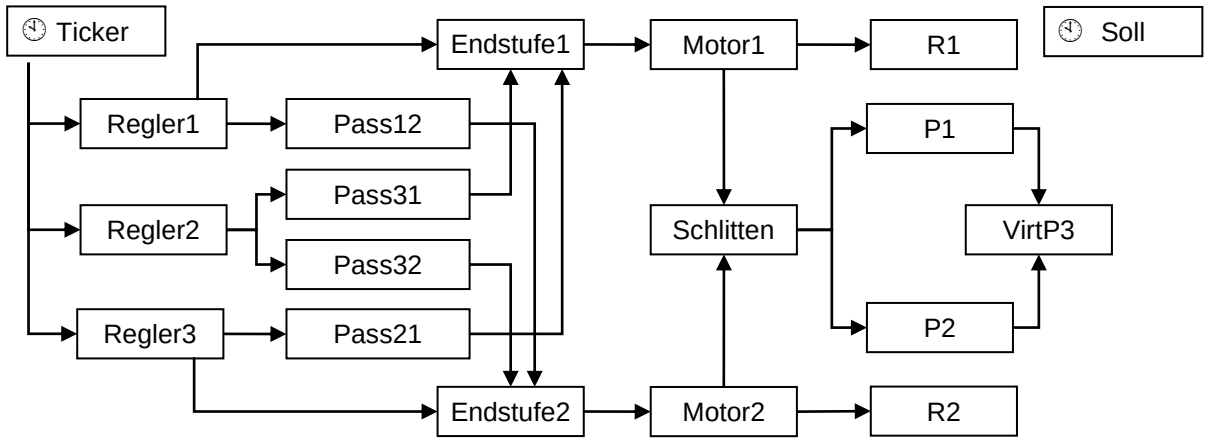


Abbildung 36: Elektronisch geregelte Lenkung, UPPAAL-Modell

Ein zeitbehafteter „Ticker“-Prozess stößt zunächst das gesamte System in zyklisch wiederkehrender Folge an und übergibt sodann die Kontrolle an die drei Prozesse Regler1, Regler2 und Regler3, in welchen die Berechnung der Regelfunktion nachgebildet ist. Über die Endstufen Endstufe1 und Endstufe2, welche über vier Passivierungsprozesse Pass12, Pass31, Pass32 und Pass21 passiviert werden können, wird der errechnete Steuerbefehl an die beiden Motoren Motor1 und Motor2 geleitet. Die sich ergebende Rotation wird vom Schlitten-Prozess in die Bewegung des Lenkgetriebes übersetzt. Sowohl die Rotation der Motoren als auch die Position des Getriebes werden über Sensoren gemessen (R1 und R2 bzw. P1 und P2). Der dritte Positionssensor ist, wie bereits im vorangegangenen Fehlerbaummodell, durch einen Softwareprozess (hier: VirtP3) ersetzt worden, da dieser Positionswert aus den Werten der Rotationssensoren berechnet wird. Schließlich induziert der zeitbehaftete Soll-Prozess eine Regeldifferenz in das System, um die korrekte Reaktion der Lenkung auf diese Störung nachweisen zu können.

Im Vergleich zur Fehlerbaumdarstellung des letzten Kapitels wird deutlich, dass einige Systemkomponenten unverändert auch in diesem Modell vorkommen, andere Elemente detaillierter dargestellt sind, manche jedoch komplett fehlen bzw. wieder andere gänzlich neu hinzugekommen sind. Ursache für diese Abweichung ist zum einen der veränderte Untersuchungsgegenstand, welcher sich vom grundsätzlichen Aufbau des Systems aus unterschiedlichen Komponenten hin zur erbrachten Systemfunktion verlagert. Des Weiteren sind an manchen Stellen praktische Aspekte bei der Modellierung von Prozessen im verwendeten Programm UPPAAL ausschlaggebend gewesen. So ist die Kommunikation über das Bussystem aus praktischen Gründen nicht mitmodelliert, die Pfeile in obiger Abbildung beschreiben also den Kontrollfluss, nicht jedoch den Datenfluss. Dieser Datenfluss findet gleichwohl statt, allerdings (wie in UPPAAL üblich) nicht über explizit modellierte Komponenten, sondern durch Nutzung gemeinsamer globaler Variablen.

Abschließend sei darauf hingewiesen, dass die im Folgenden dargestellten Prozesse zur besseren Lesbarkeit ausschließlich den fehlerfreien Fall beschreiben. Zur Verifikation des korrekten Verhaltens im Fehlerfall entstand eine zusätzliche Modellvariante mit Fehlerinjektion. Ihr grundsätzlicher Aufbau sowie die Modellierungstechnik sind am Ende dieses Kapitels beschrieben. Das komplette Modell (einschl. Fehlerinjektor) ist im elektronischen Anhang der Arbeit enthalten, die Deklaration der globalen Variablen befindet sich in Anhang A.

Der Prozess `TickerT` (s. Abbildung 37) wartet zunächst im Zustand `passiv` auf das Verstreichen der durch die Konstante `ZyklusD` vorgegebenen Zyklusdauer. Sobald diese erreicht ist `(tZyklus == ZyklusD)`, wird die entsprechende Uhr zurückgesetzt, ein Zykluszähler erhöht und allen nachgelagerten Prozessen der Beginn des neuen Zyklus über den Kanal `TickerK` signalisiert.

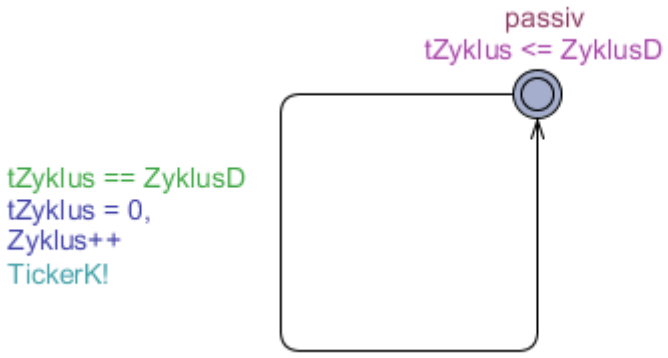


Abbildung 37: Prozess TickerT

Im so angestoßenen Prozess `RechnerReglerT` (Abbildung 38) findet die Berechnung der Regelfunktion statt. Für jede der drei ECUs der elektronischen Lenkung existiert eine eigene Instanz dieses Prozesses, in Abbildung 36 mit `Regler1`, `Regler2` bzw. `Regler3` bezeichnet und im Modell durch den Parameter `i` unterschieden.

Das Verhalten ist für alle drei Instanzen identisch: Im Anschluss an die Synchronisation mit dem `Ticker`-Prozess wird die Regeldifferenz zwischen dem Istwert des der entsprechenden ECU zugeordneten Positionssensors `PosW[i]` und dem globalen Sollwert `SollW` berechnet. Nach einer durch die Konstante `ReglerD` vorgegebenen Latenz wird das aus dieser Differenz resultierende Steuerkommando `ReglerW[i]` berechnet. Die Werte `-1`, `0` und `1` stehen dabei für die drei möglichen Kommandos „links“, „geradeaus“ und „rechts“. Abschließend wird nachfolgenden Prozessen die Verfügbarkeit des Steuerbefehls über den Kanal `ReglerK[i]` signalisiert.

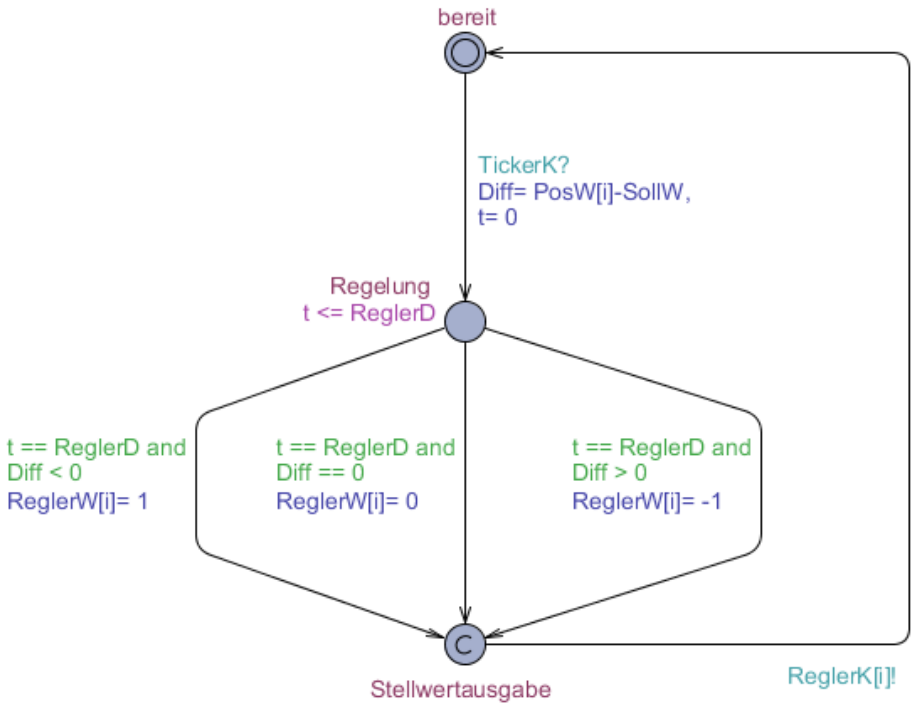


Abbildung 38: Prozess RechnerReglerT

Wie in Abbildung 36 ersichtlich, existieren insgesamt vier Passivierungsprozesse `Pass12`, `Pass31`, `Pass32` und `Pass21`. Durch die Bezeichnungen werden die bereits aus vorhergehenden Erläuterungen bekannten Zuordnungen ausgedrückt: ECU_1 passiviert Motor M2, ECU_3 passiviert die Motoren M_1 und M_2, ECU_2 passiviert Motor M_1. Alle vier Prozessinstanzen sind abgeleitet von `RechnerPassT` mit den beiden Parametern `i` (passivierende ECU) und `j` (zu passivierender Motor). Gültige Parameterkombinationen für `i` und `j` sind somit die Tupel $(1, 2), (3, 1), (3, 2)$ und $(2, 1)$.

Auch hier ist der Ablauf für jede Instanz identisch (s. Abbildung 39): Das lokale Steuerkommando `ReglerW[i]` wird zunächst in der Variablen `eigen` gespeichert. Nach Ablauf der durch `PruefD` ausgedrückten Zeitspanne wird die Rotation des ggf. zu passivierenden Motors `RotW[j]` ausgelesen und in der Variablen `fremd` gespeichert. Wenn nun diese beiden Werte voneinander abweichen oder die empfangene Rotation nicht korrekt signiert war, wird in `PassW[i][j]` vorgemerkt, dass der betreffende Motor passiviert werden soll. Jedes Passivierungskommando wird mit einer Signatur `PassS[i][j]` versehen, während der Prozess selbst zurück in den Zustand `bereit` wechselt, die Variablen `eigen` und `fremd` zurücksetzt und das Vorliegen des Passivierungskommandos über den Kanal `PassK[i][j]` signalisiert.

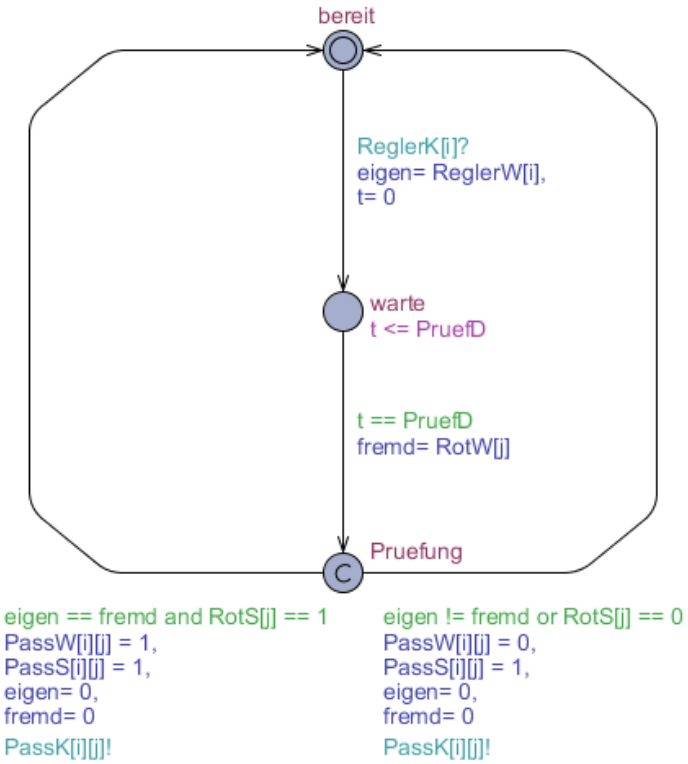


Abbildung 39: Prozess RechnerPassT

Die beiden in Abbildung 36 als `Endstufe1` und `Endstufe2` bezeichneten Komponenten sind Instanzen des Prozesses `EndstufeT` (s. Abbildung 40). Dieser Prozess ist parametrisiert mit den Parametern `i`, `j` und `k`, welche ausdrücken, dass Motor `i` gemeinsam von ECUs `j` und `k` passiviert wird. Da keine ECU den von ihr selbst angesteuerten Motor passivieren darf, sind die gültigen Parameterkombinationen auf die Tupel $(1, 2, 3)$ und $(2, 1, 3)$ beschränkt: Motor M_1 wird gemeinsam von ECU_2 und ECU_3 passiviert, Motor M_2 gemeinsam von ECU_1 und ECU_3.

Für beide sich so ergebenden Instanzen ist auch hier der Ablauf identisch: Ausgehend vom Zustand `bereit` wird bei Vorliegen eines neuen Steuerbefehls (Synchronisation mit `ReglerK[i]`) dieser Steuerbefehl `ReglerW[i]` in die Variable `EndstufeW[i]` übernommen. Die Passivierungsprozesse können in beliebiger Reihenfolge (Synchronisation mit `PassK[j][i]` und `PassK[k][i]`) das Vorliegen neuer Passivierungskommandos signalisieren. Wenn mindestens einer dieser Prozesse die Passivierung nicht für notwendig hält (`PassW[j][i]` bzw. `PassW[k][i]`) und dieses Signal korrekt signiert ist (`PassS[j][i]` bzw. `PassS[k][i]`), bleibt die bereits in `EndstufeW[i]` vorgemerkte Ansteuerung bestehen und der Prozess kehrt zurück in den Zustand `bereit`. Gilt dies nicht,

so stimmen entweder beide Prozesse für eine Passivierung (`not PassW[j][i]` bzw. `not Pass W[k][i]`) oder es liegen nicht genügend gültige Aktivierungssignale vor (im Modell nachgebildet durch eine ungültige Signatur (`not PassS[j][i]` bzw. `not PassS[k][i]`). In diesem Fall wird durch die Zuweisung `EndstufeW[i] = 0` und den Wechsel in den Zustand `passiviert` erreicht, dass der betreffende Motor fortan inaktiv bleibt. Nachfolgende Prozesse werden über den Kanal `EndstufeK[i]` über den Abschluss dieses Vorgangs benachrichtigt.

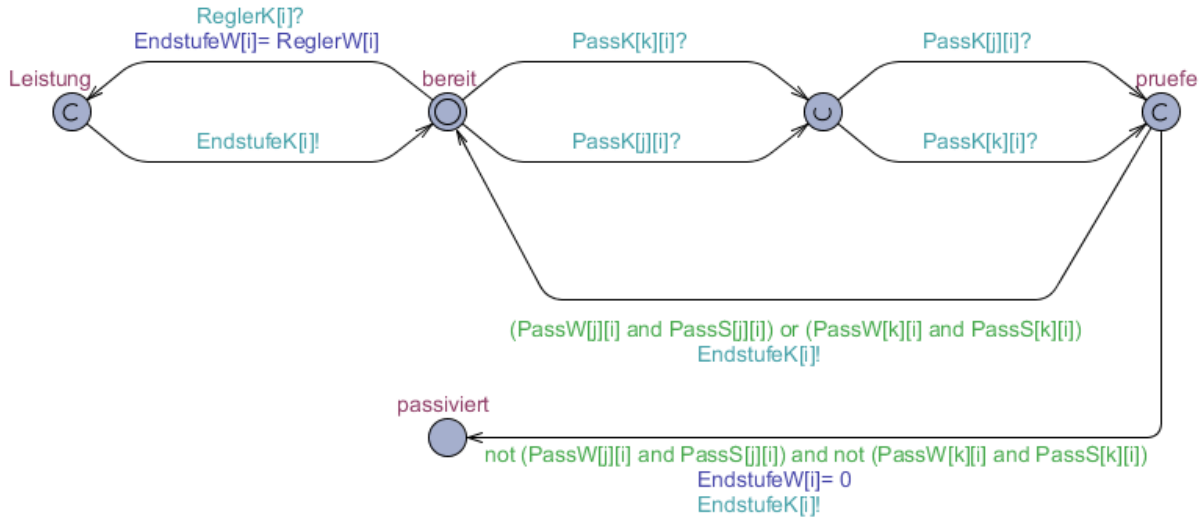


Abbildung 40: Prozess EndstufeT

Die beiden Motoren `Motor1` und `Motor2` aus Abbildung 36 sind Instanzen des Prozesses `MotorT` (s. Abbildung 41). Einziger Parameter ist die Bezeichnung des betreffenden Motors, `i`.

Beide Motoren speichern bei Synchronisierung mit der zugehörigen Endstufe über `EndstufeK[i]` zunächst den von dieser weitergeleiteten und nun in `EndstufeW[i]` vorliegenden Steuerbefehl in der lokalen Variablen `Richtung`. Falls keine Richtungsänderung vorliegt (`Richtung == MotorW[i]`) kehrt der Prozess in seinen Ausgangszustand `unveraendert` zurück. Andernfalls (`Richtung != MotorW[i]`) wartet der Prozess im Zustand `vorAenderung`, bis die durch `MotorD` vorgegebene Zeit verstreicht. Ein zwischenzeitlicher Wechsel des Richtungskommandos ist möglich und wird durch Übergang in den Zustand `neueVorgabe` verarbeitet. Die Richtungsänderung selbst (Zustand `Aenderung`) wird modelliert, indem die Motorrotation erhöht (`MotorW[i]++`) bzw.

verringert (`MotorW[i]--`) wird. Das letztendliche Vorliegen einer Richtungsänderung wird den nachfolgenden Prozessen über den Kanal `MotorK` signalisiert.

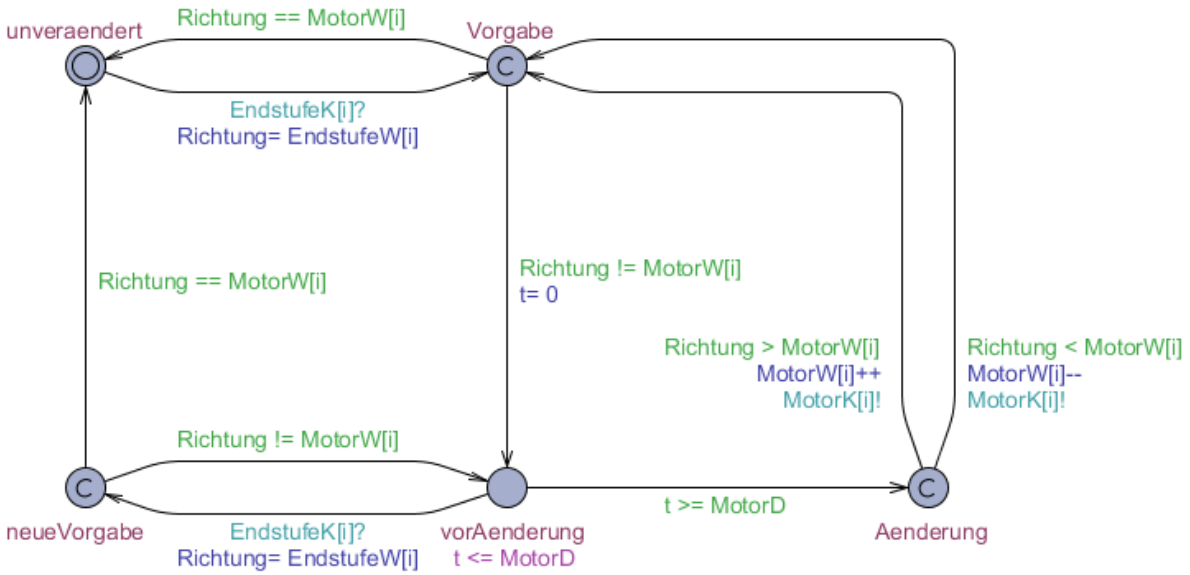


Abbildung 41: Prozess MotorT

Die in Abbildung 36 mit `R1` und `R2` bezeichneten, die Motoren überwachenden Rotationssensoren sind Instanzen des Prozesses `SensorRotT` (s. Abbildung 42) und werden über den Parameter `i` einem Knoten zugeordnet.

Das Verhalten der Rotationssensoren besteht darin, dass, ausgelöst durch Synchronisation mit `MotorK[i]`, die aktuelle Rotation des betreffenden Motors in der Variablen `RotW[i]` gespeichert und sodann mit einer gültigen Signatur versehen wird (`RotS[i] = 1`).

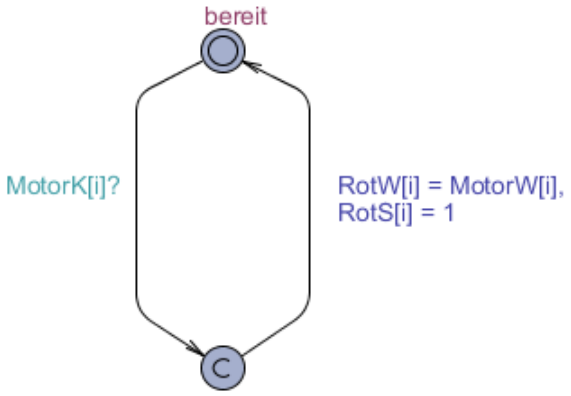


Abbildung 42: Prozess SensorRotT

Der die im Perfektionskern befindliche Mechanik der Lenkung abbildende Prozess `Schlitten` aus Abbildung 36 ist eine Instanz von `SchlittenT` (vgl. Abbildung 43) und wird nicht parametrisiert.

Ausgehend vom Zustand `Stillstand` wechselt dieser Prozess durch Synchronisation mit `MotorK[1]` oder `MotorK[2]` in den Zustand `Vorgabe`. Wenn keiner der beiden Motoren sich bewegt oder die Motoren sich gegenläufig drehen (`MotorW[1] + MotorW[2] == 0`), kehrt der Prozess in seinen Ausgangszustand zurück. Andernfalls wird nach Ablauf der durch `SchlittenD` vorgegebenen Zeit der Zustand `neuePosition` erreicht. Je nach Richtung der Änderung (`MotorW[1] + MotorW[2] > 0` bzw. `MotorW[1] + MotorW[2] < 0`) wird der Wert für die Position des Schlittens erhöht bzw. verringert (`SchlittenW++` bzw. `SchlittenW--`), sofern die durch `minR` und `maxR` vorgegebene maximale Auslenkung dabei nicht erreicht wird. Den nachfolgenden Prozessen wird diese Änderung über den Kanal `SensorK` signalisiert, während der Prozess zunächst in den Zustand `Bewegung` zurückkehrt. Sobald keine Bewegung mehr stattfindet (`MotorW[1] + MotorW[2] == 0`), erreicht der Prozess den Zustand `Stillstand`. Bei einem Überschreiten der maximalen Auslenkung wechselt der Prozess in den Zustand `amAnschlag` und ermöglicht es so, zu erkennen, dass der Schlitten aufgrund einer Fehlfunktion zu weit bewegt wurde.

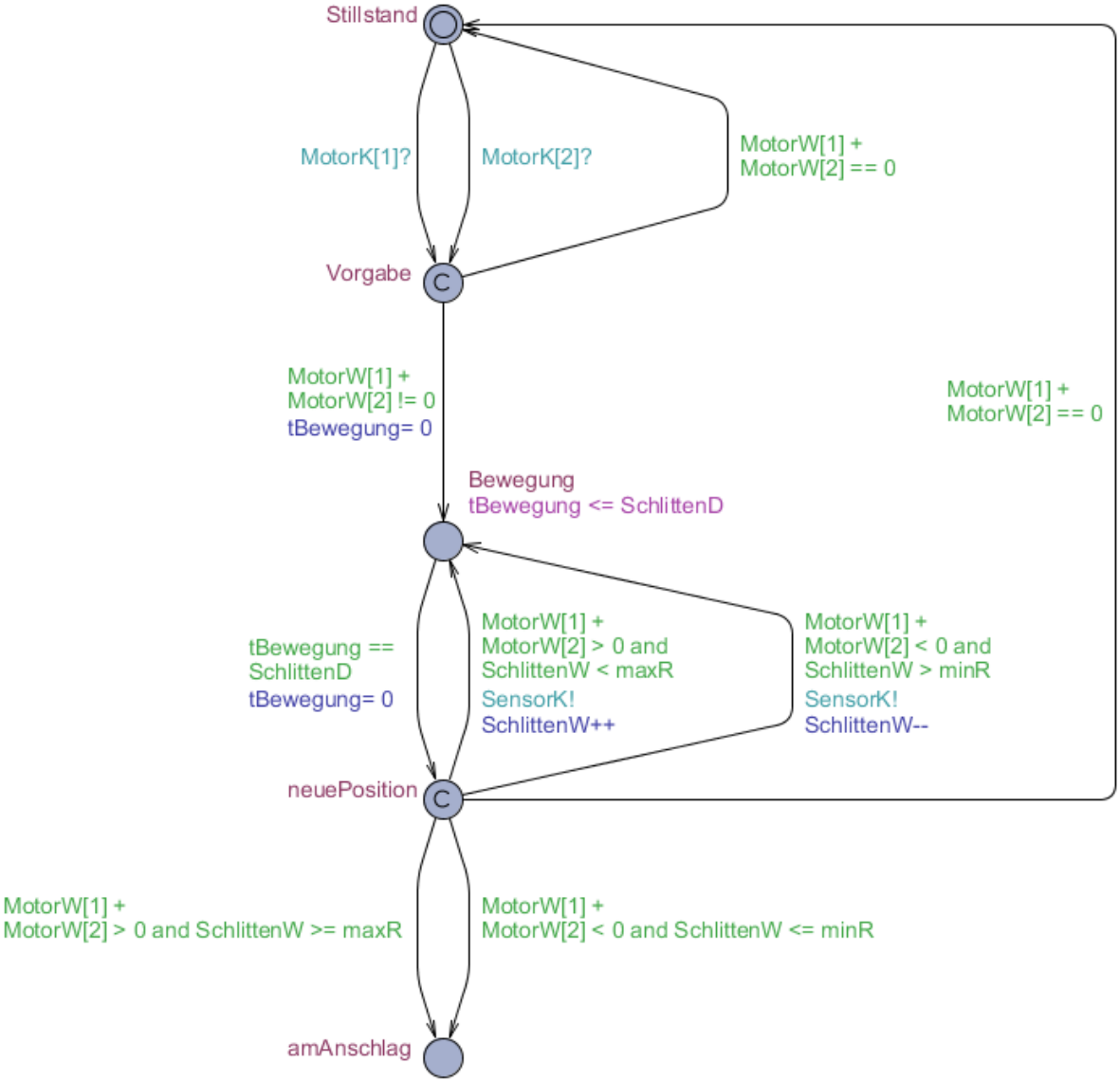


Abbildung 43: Prozess SchlittenT

Die beiden Positionssensoren `P1` und `P2` aus Abbildung 36 sind Instanzen von `SensorPosT` (s. Abbildung 44) und werden mit dem Parameter `i` dem zugehörigen Knoten zugeordnet.

Nach Synchronisation mit `SensorK` (ausgelöst durch eine Bewegung des Schlittens) übernehmen sie dessen Position in `PosW[i]`, signieren diesen Wert (`PosS[i] = 1`) und signalisieren das Vorliegen einer neuen Position über `RechnerSensK`.

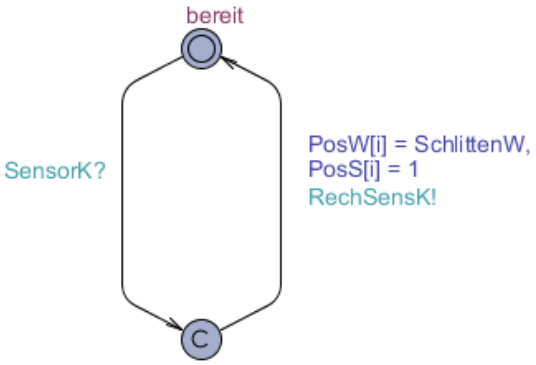


Abbildung 44: Prozess SensorPosT

Der dritte, virtuelle Positionssensor `VirtP3` aus Abbildung 36 ist eine Instanz des Prozesses `RechnerSensT` (s. Abbildung 45) und wird nicht parametrisiert.

Der Prozess benutzt die Rotationssensoren als Referenz, um bei einer Abweichung zwischen beiden Positionssensoren den jeweils richtigen Wert zu ermitteln. Diese Berechnung wird jedes Mal ausgelöst, wenn die Positionssensoren neue Werte liefern. Wenn nun eine der beiden Signaturen ungültig ist, wird der jeweils andere (nach Annahme korrekt arbeitende) Sensor als `PosW[3]` übernommen. Wenn P_1 und P_2 übereinstimmen, wird willkürlich P_1 gewählt. Nur wenn beide Sensoren in ihrem Wert voneinander abweichen, wird die Funktion `calcP3()` aufgerufen, um eine Näherung für P_3 auf Basis der Rotation der beiden Motoren zu berechnen und den Positionssensor zu wählen, welcher dieser Näherung am nächsten liegt.

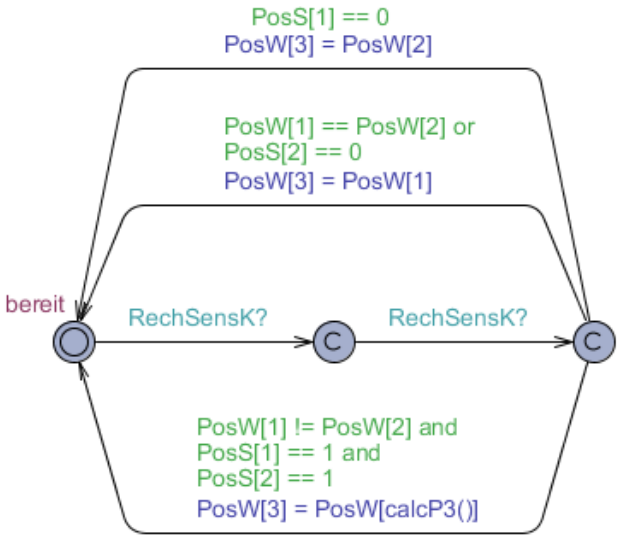


Abbildung 45: Prozess RechnerSensT

Die nachfolgende Abbildung 46 enthält den Programmcode der Funktion `calcP3` für die Ermittlung des gültigen Positionssensors: Zunächst wird in der Variablen `p3n` eine Näherung aus den Richtungsinformationen beider Rotationssensoren berechnet. Zurückgeliefert wird dann derjenige Positionssensor, dessen Wert der Näherung am nächsten liegt. Die ebenfalls abgebildeten Funktionen `abs` und `richtung` sind Hilfsfunktionen, welche aus einem übergebenen Wert seinen Absolutwert bzw. seine Richtungsinformation (`1`, `-1` oder `0`) berechnen.

```
int abs(int wert) {
    if (wert >= 0)
        return wert;
    else
        return -wert;
}

int[-1,1] richtung(int wert) {
    if (wert > 0)
       return 1;
    else if (wert < 0 )
       return -1;
    else
       return 0;
}

int[1,2] calcP3() {
    int[minR-2,maxR+2] p3n = PosW[3] + richtung(RotW[1] + RotW[2]);
    if (abs(p3n-PosW[1]) > abs(p3n-PosW[2]))
        return 2;
    else
        return 1;
}
```

Abbildung 46: Näherungsweise Berechnung des dritten Positionswertes

Der Prozess `Soll` aus Abbildung 36 schließlich ist eine Instanz von `VorgabeT` (s. Abbildung 47) und wird ebenfalls nicht parametrisiert. Der Prozess wartet im Ausgangszustand auf das Verstreichen der Zeitspanne `DiffD`. Ist diese erreicht, so wird die Sollvorgabe `SollW` abwechselnd zwischen den Werten `diffR` und `0` umgeschaltet. Die so induzierte neue Richtungsvorgabe ermöglicht es, die hierdurch ausgelöste Reaktion der Lenkung (im korrekten Fall also die fortschreitende Verringerung der Regeldifferenz) zu

beobachten. Zugleich wird die Variable `Zyklus` zurückgesetzt, so dass es möglich ist, die für eine erfolgreiche Ausregelung benötigte Zeitdauer zu ermitteln.

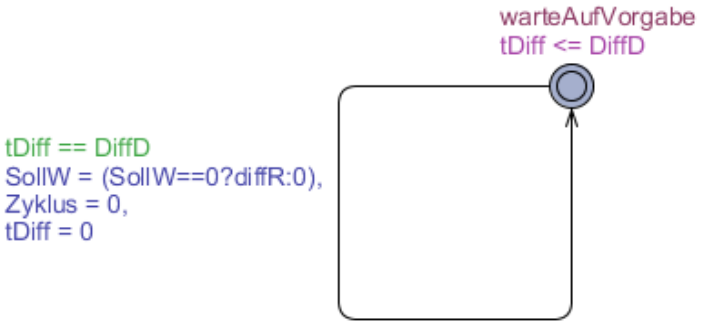


Abbildung 47: Prozess VorgabeT

Das hiermit vollständige Modell der elektronisch geregelten Lenkung wurde für die nachfolgende Untersuchung um die Möglichkeit ergänzt, gezielt Fehler zu injizieren, so dass die korrekte Funktion der Lenkung nicht nur im fehlerfreien Fall, sondern auch bei Vorliegen bestimmter Fehler überprüft werden kann. Die bereits für die Fehlerbaumanalyse in Kapitel 5.2.1 beschriebene Einfehlerannahme wurde hierbei grundsätzlich beibehalten, jedoch in Form folgender Fehlermöglichkeiten weiter präzisiert:

- Alle „entfernt redundant“ für einen anderen Knoten operierenden Prozesse (d. h. Positionssensoren, Rotationssensoren und Passivierungsprozesse) können sowohl hinsichtlich der von ihnen gelieferten Werte als auch bezüglich der bei der Übermittlung verwendeten Signaturen Fehler aufweisen.
- Bei nur lokal operierenden Prozessen (Reglungsfunktion und „virtueller“ Positionssensor P_3) sind nur Wertefehler relevant, Signaturen werden hier nicht verwendet.
- Motoren können im Fehlerfall ausschließlich anhalten, da sie selbst keine Energie erzeugen können. Ein Ansteuern in die falsche Richtung wird als Fehler der zugehörigen Regelfunktion modelliert.
- Endstufen können den an sie angeschlossenen Motor beliebig aktivieren oder passivieren, wobei aufgrund der Einfehlerannahme lediglich die Passivierung Auswirkungen hat.

Die Markierung von Komponenten als fehlerhaft ist im Modell durch globale Konstanten realisiert. So gibt etwa `fiRotW = 2` an, dass die Werte des zweiten Rotationssensors fehlerhaft sind. Zuordnungen innerhalb von Prozessen wie `RotW[i] = MotorW[i]` werden in der Modellvariante mit Fehlerinjektion ersetzt durch `RotW[i] = (i != fiRotW?MotorW[i]:fiWert)`. Hiermit wird die grundsätzliche Möglichkeit geschaffen, fehlerhafte Daten zu injizieren. Der tatsächliche Wert der zu injizierenden fehlerhaften Daten kann entweder explizit (als Konstante) festgelegt werden oder über indeterministische Prozesse zur Laufzeit verändert werden. Die erste Möglichkeit wird über einfache Zuweisungen (etwa `fiWert = -1`) realisiert, die zweite Möglichkeit ist in Abbildung 48 dargestellt. Der zulässige Wertebereich der der Fehlerinjektion dienenden Variablen hängt davon ab, ob es sich um eine Signatur, eine Richtungsinformation oder einen Postitionswert handelt, wobei Verspätung, Verlust oder Verfälschung von Bus-Nachrichten als Signaturfehler nachgebildet werden.

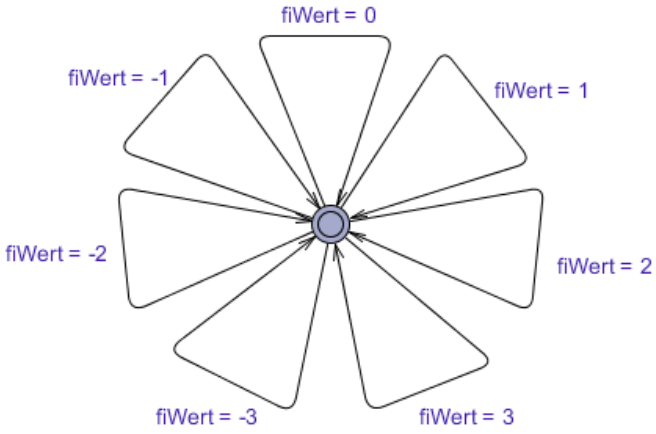


Abbildung 48: Indeterministische Wertzuweisung zur Fehlerinjektion

5.2.2.3 Ergebnis

Zur Prüfung der Fehlertoleranzeigenschaften des modellierten Systems sind vorab sog. „Properties“ zu definieren, welche das gewünschte Verhalten in der von UPPAAL vorgegebenen Syntax ausdrücken. Die im Rahmen dieser Arbeit verwendeten Properties sind in nachfolgender Tabelle 6 dargestellt.

Als wichtigste Eigenschaft ist zunächst sicherzustellen, dass die geforderte Fehlertoleranz erreicht wird, mit anderen Worten also, dass das System das spezifizierte Verhalten auch bei Auftreten der vorgegebenen Fehler zeigt. Dies wird für das Beispielsystem for-

muliert als: „Höchstens n Zyklen nach dem Induzieren der Regeldifferenz wird diese Differenz bis auf eine bestimmte Toleranz reduziert“ (Property `P1`). Diese Property reicht bereits aus, um die geforderte Fehlertoleranz nachzuweisen. Zusätzliche Properties wurden jedoch eingeführt, um das Verhalten des Systems genauer zu beobachten, d. h. insbesondere die Passivierung der Motoren zu überwachen (Properties `P2` bis `P5`). Diese Properties implizieren für sich genommen zwar noch kein fehlertolerantes Verhalten, ermöglichen es jedoch, nachzuverfolgen, ob beim Auftreten von Fehlern die entsprechenden Gegenmaßnahmen wie erwartet durchgeführt werden bzw. ob diese im fehlerfreien Fall tatsächlich ausbleiben. Die Properties `P6` und `P7` sind weitere Plausibilitätstests, welche nachweisen, dass das simulierte Lenkgetriebe nicht den Anschlag der nachgebildeten Mechanik erreicht und Überschwinger innerhalb einer vorgegebenen Toleranz bleiben. Properties `P8` und `P9` garantieren, dass Regeldifferenzen induziert werden und verifizieren somit die Versuchsanordnung selbst. Property `P10` schließlich stellt sicher, dass das System frei von Verklemmungen ist.

Tabelle 6: Zur Verifikation ausgewählte Systemeigenschaften

Nr.	Eigenschaft	Beschreibung
P1	`A[] (Zyklus >= 6 imply (SchlittenW-SollW >= -2 and SchlittenW-SollW <= 2))`	Nach jeder Ausregelung ist ab Zyklus 6 die Regeldifferenz höchstens 2.
P2	`A<> Endstufe1.passiviert`	Endstufe 1 wird in jedem Fall passiviert.
P3	`A[] not Endstufe2.passiviert`	Endstufe 2 wird niemals passiviert.
P4	`A<> Endstufe2.passiviert`	Endstufe 2 wird in jedem Fall passiviert.
P5	`A[] not Endstufe1.passiviert`	Endstufe 1 wird niemals passiviert.
P6	`A[] not Schlitten.amAnschlag`	Der Schlitten befindet sich niemals am Anschlag.
P7	`A[] (SchlittenW >= -2 and SchlittenW-diffR <= 2)`	Überschwinger nach oben oder unten um höchstens 2.
P8	`SchlittenW >= diffR --> SchlittenW >= diffR`	Auf jede Auslenkung folgt eine weitere.
P9	`A<> SchlittenW >= diffR`	Es gibt immer eine erste Auslenkung.
P10	`A[] not deadlock`	Das System ist niemals verklemmt.

Tabelle 7 enthält die im Verlauf der Arbeit durch Prüfung obiger Properties gewonnenen Verifikationsergebnisse. Wie aus der Tabelle ersichtlich, konnte dabei gezeigt werden, dass `P1` tatsächlich immer gilt, d. h. dass das System seine Spezifikation bei allen vorgegebenen Einzelfehlern erfüllt (sowie selbstverständlich im fehlerfreien Fall). Ferner gelang es, nachzuweisen, dass auch die Properties `P6` bis `P10` immer gelten. Alle

Plausibilitätstests, das Systemverhalten betreffend, sind also stets erfüllt. Die Properties `P2` bis `P5` gelten erwartungsgemäß abhängig vom Fehlerort: Stets wird der vom Fehler betroffene Motor passiviert, während der nicht betroffene Motor weiterhin seine Funktion erbringt.

Tabelle 7: Verifikationsergebnisse

Komponente	Fehlerart	Eigenschaft									
		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
(keine)	`(fehlerfreier Fall)`	✓	×	✓	×	✓	✓	✓	✓	✓	✓
SensorRot1/2	`Wertefehler`	✓	✓/×	✓/×	×/✓	×/✓	✓	✓	✓	✓	✓
	`Signaturfehler`	✓	✓/×	✓/×	×/✓	×/✓	✓	✓	✓	✓	✓
SensorPos1/2	`Wertefehler`	✓	✓/×	✓/×	×/✓	×/✓	✓	✓	✓	✓	✓
	`Signaturfehler`	✓	×	✓	×	✓	✓	✓	✓	✓	✓
Rechner Sens	`Wertefehler`	✓	×	✓	×	✓	✓	✓	✓	✓	✓
Rechner Regler1/2/3	`Wertefehler`	✓	✓/×/×	✓/×/✓	×/✓/×	×/✓/✓	✓	✓	✓	✓	✓
Rechner Pass12/3x/21	`Wertefehler`	✓	×	✓	×	✓	✓	✓	✓	✓	✓
	`Signaturfehler`	✓	×	✓	×	✓	✓	✓	✓	✓	✓
Endstufe1/2	`Aus`	✓	✓/×	✓/×	×/✓	×/✓	✓	✓	✓	✓	✓
Motor1/2	`Stopp`	✓	✓/×	✓/×	×/✓	×/✓	✓	✓	✓	✓	✓

Im Detail kann das verifizierte Verhalten wie folgt beschrieben werden: Im fehlerfreien Fall wird wie erwartet niemals passiviert (`P3` und `P5` sind erfüllt, `P2` und `P4` jedoch nicht). Im Falle eines fehlerhaften Rotationssensors wird sowohl bei falschen wie auch bei ungültig signierten Werten die betroffene Endstufe passiviert, die jeweils andere bleibt in Betrieb (entweder `P2` und `P3` oder `P4` und `P5` sind erfüllt). Wertefehler der Positionssensoren führen zu einer falschen Ansteuerung des zugehörigen Motors und folglich ebenso zur Passivierung der entsprechenden Endstufe, jedoch nicht der jeweils anderen. Positionsdaten mit einer falschen Signatur werden bei der Berechnung des virtuellen Positionssensors ignoriert (`P3` und `P5` sind erfüllt, `P2` und `P4` nicht). Knoten 3 kann alleine keine Passivierung vornehmen, daher ist eine vom virtuellen Positionssensor verursachte oder vom entsprechenden Regelprozess getroffene falsche Regelentscheidung (wenn kein weiterer Fehler auftritt) irrelevant (`P3` und `P5` sind erfüllt, `P2` und `P4` hingegen nicht). Bei einer falschen Regelentscheidung in Knoten 1 oder 2 wird der zugehörige Motor falsch angesteuert und daher

korrekterweise passiviert, der andere jedoch erwartungsgemäß nicht (entweder P2 und P3 oder P4 und P5 sind erfüllt). Auch für Fehler bei der Passivierung gilt (unabhängig von der Art des Fehlers): Da ein Rechner alleine nicht passivieren kann, bleiben Einzelfehler ohne Wirkung (P3 und P5 sind erfüllt, P2 und P4 nicht). Bei einem Ausfall der Endstufe wird der zugehörige Motor passiviert, der jeweils andere jedoch nicht. Auch bei einem Stillstand eines Motors wird dieser passiviert, der bleibt in Betrieb (für beide Fehlerarten gilt: entweder P2 und P3 oder P4 und P5 sind erfüllt).

Das oben beschriebene Verhalten im Fehlerfall betrifft jeweils sog. „Stuck at"-Fehler *(dt.: Haftfehler)*, wobei der fehlerhafte Wert für die Dauer des Fehlers unverändert bleibt. Der dabei im Rahmen der vorliegenden Arbeit untersuchte Wertebereich betraf für eine Richtungsangabe: $-1, \ldots, 1$, für eine Positionsangabe: $-3, \ldots, 3$, für eine Passivierungsaufforderung: 0, für eine fehlerhafte Signatur: 0, für eine defekte Endstufe: „aus", für einen defekten Motor: „Stopp". Auch für beliebig im Zeitverlauf wechselnde, indeterministisch zugewiesene, fehlerhafte Werte konnte die korrekte Lenkfunktion (Property P1) stets verifiziert werden. Da bei der Betrachtung aller möglichen Werteverläufe jedoch stets der fehlerfreie Fall und/oder unproblematische Verläufe mit eingeschlossen sind, gelten die P2 bis P5 bei dieser Fehlerart nicht grundsätzlich, da die Notwendigkeit zur Passivierung nicht in jedem Fall besteht und somit nicht generell eingefordert werden kann.

Die Log-Dateien der hier beschriebenen Abfragen sowie die Abfragen selbst sind im elektronischen Anhang der Arbeit vollständig enthalten.

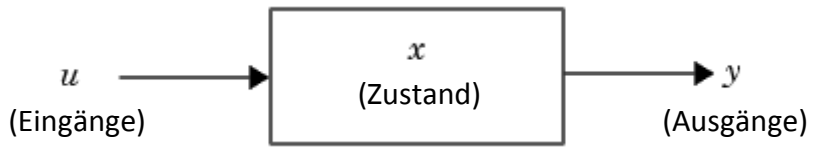
Insgesamt konnte durch die Verifikationsergebnisse in diesem Teil der Arbeit ein wichtiges Resultat der Fehlerbaumanalyse bestätigt werden: alle im Fehlermodell enthaltenen Einzelfehler werden wie gefordert toleriert. Eine Untersuchung der Tolerierung von Doppelfehlern oder ein Vergleich mit Dedizierter Redundanz war hier nicht Gegenstand der Untersuchung. Es konnte jedoch erstmals formal nachgewiesen werden, dass das Beispielsystem der elektronisch geregelten Lenkung mit Entfernter Redundanz in einem noch relativ groben, aber voll funktionsfähigen Modell fehlertolerant arbeitet.

5.2.3 Funktionale Simulation

5.2.3.1 Methode

Wie bereits oben erwähnt, musste das im vorangegangenen Kapitel erläuterte Modell, auch wenn es bereits die Lenkfunktion des untersuchten Systems nachbildet, aus Gründen der Handhabbarkeit relativ undetailliert bleiben: Zum einen geben die im verwendeten Programm verfügbaren Datentypen Einschränkungen vor, zum anderen muss die Größe des Zustandsraumes für eine formale Verifikation notwendigerweise beschränkt bleiben. Für eine genauere Betrachtung des Systemverhaltens ist es also unumgänglich, diese Ebene zu verlassen. Das kommerzielle Programmpaket MATLAB[12] bietet mit seinem Simulationswerkzeug Simulink die Möglichkeit, dynamische Systeme mit einem sehr hohen Detaillierungsgrad zu modellieren, zu simulieren und zu analysieren. Als Ergänzung zur vollständigen Verifikation eines relativ abstrakten Modells wurde daher mit seiner Hilfe im Anschluss eine (allerdings zwingend stichprobenartige) Untersuchung eines wesentlich genaueren Modells der elektronisch geregelten Lenkung durchgeführt. Bevor dieses Modell erläutert wird, soll auch hier zunächst die in diesem Teil der Arbeit verwendete Modellwelt genauer vorgestellt werden.

Simulink-Modelle bestehen aus miteinander verbundenen Blöcken, welche im Zusammenspiel das Systemverhalten beschreiben. Jeder Block steht für einen Teil des Systems und kann entweder kontinuierlich oder diskret zu bestimmten Zeiten Ausgaben erzeugen. Ein solcher Block (s. Abbildung 49) besteht stets aus einer Menge von Eingängen *u*, einem Zustand *x* und Ausgängen *y*. Der Zusammenhang zwischen diesen Größen in Abhängigkeit von der Zeit wird bestimmt vom jeweiligen Typ des Blocks, d. h. seinem vorgegebenen Verhalten.



vgl. (The MathWorks, 2002)

Abbildung 49: Simulink-Block

[12] http://www.mathworks.de/products/matlab/

Das Verhalten der ein System beschreibenden Blöcke wird vom Benutzer spezifiziert (unter Verwendung bereits in Simulink vordefinierter Blöcke oder durch eigenen Programmcode als sog. „S-Functions“) und intern beschrieben durch eine Menge von Systemfunktionen (vgl. Abbildung 50). Hierbei ist erneut *t* die gegenwärtige Zeit, *x* der Zustand des Blocks, *u* sind die Eingänge, *y* die Ausgänge. Der Zustand *x* wird aufgeteilt in einen diskreten Anteil x_d und einen kontinuierlichen Anteil x_c.

$$y = f_o(t, x, u) \qquad \text{Ausgabefunktion}$$

$$x_{d_{k+1}} = f_u(t, x, u) \qquad \text{Aktualisierungsfunktion}$$

$$x'_c = f_d(t, x, u) \qquad \text{Ableitungsfunktion}$$

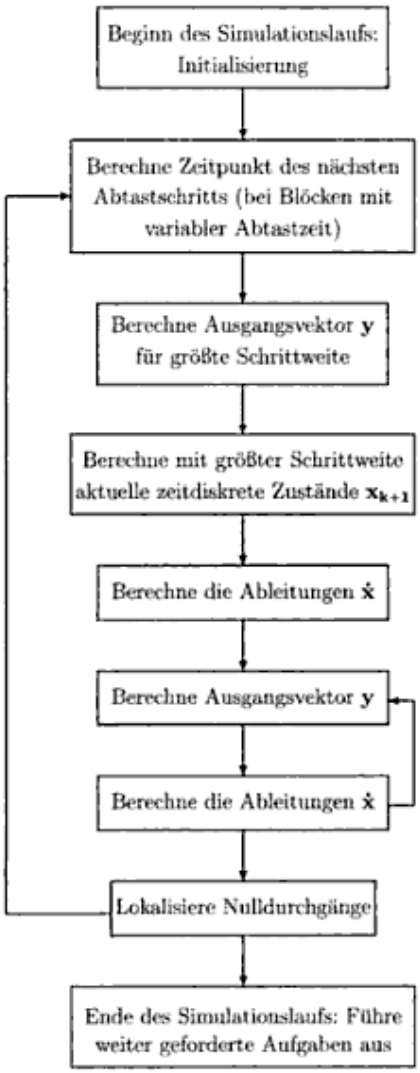
$$\text{mit} \quad x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

vgl. (The MathWorks, 2002)

Abbildung 50: MATLAB/Simulink-Systemfunktionen

Während der Simulation werden nun zyklisch die oben aufgeführten Funktionen aufgerufen, um die jeweils nächsten Zustände und die Ausgaben des Systems zu berechnen: Die Ausgabefunktion f_o berechnet die Ausgaben des Systems in Abhängigkeit von Zeit, dem Zustand und den Eingaben. Die Aktualisierungsfunktion f_u berechnet aus denselben Größen die zukünftigen Werte des diskreten Anteils des Systemzustands, die Ableitungsfunktion f_d berechnet die Änderung des kontinuierlichen Anteils des Systemzustands. Der gesamte Ablauf der Simulation stellt sich damit wie in nachfolgender Abbildung 51 beschrieben dar.

Die erstmalige Berechnung der Systemzustände während der Initialisierungsphase sowie ihre Ausgabe am Ende des Simulationslaufes finden einmalig statt. In der dazwischenliegenden Simulationsschleife werden die jeweiligen Zustände in Abhängigkeit von der eingestellten Abtastzeit und (zur Erhöhung der Genauigkeit) ggf. entsprechend der sich aus dem System selbst ergebenden Eigenschaften durch Aufruf der Systemfunktionen aktualisiert.



modif. übern. aus (Angermann, Beuschel, Rau, & Wohlfarth, 2007)

Abbildung 51: Ablauf einer Simulation mit MATLAB/Simulink

Als Beispiel für die typische Verwendung von Simulink sei abschließend ein einfaches System aus (The MathWorks, 2002) aufgeführt, welches eine Sinuskurve zusammen mit ihrem Integral im Zeitverlauf berechnet und ausgibt (s. Abbildung 52).

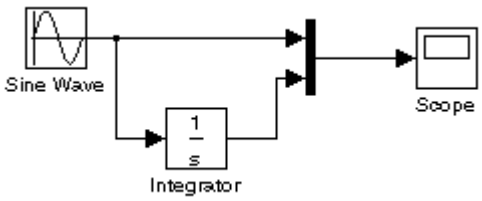


Abbildung 52: Beispielsystem mit Sinusfunktion

Das oben abgebildete System erzeugt mit der Sinuskurve und ihrem Integral zwei Signale im Zeitverlauf, welche von einem Ausgabeblock wie in Abbildung 53 dargestellt gemeinsam zu Kontrollzwecken angezeigt werden.

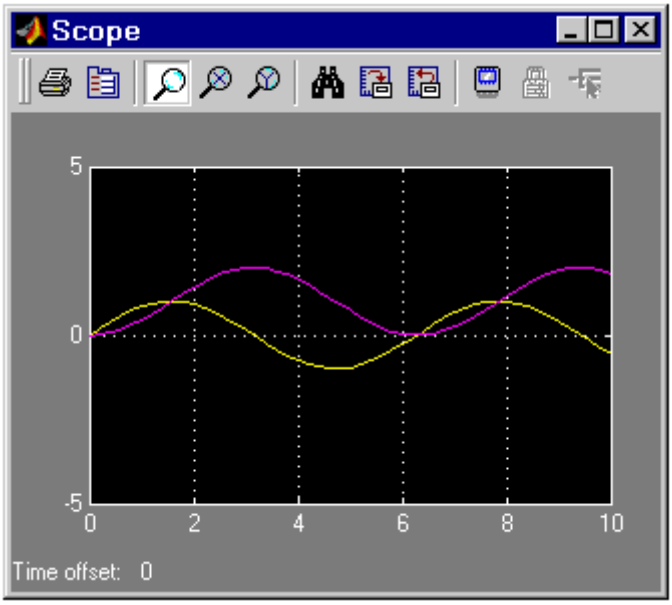


Abbildung 53: Ausgabe des Beispielsystems

Im vorliegenden Modell wird die Menge der verfügbaren Simulationsblöcke über den von MATLAB bereits mitgelieferten Bestand hinaus erweitert durch ein spezielles Toolset der Firma DECOMSYS (nunmehr Elektrobit), welches die Funktion des FlexRay-Datenbusses[13] nachbildet. Auf diese Weise konnte der Abstraktionsgrad nicht nur in Bezug auf das Verhalten des Modells verringert werden, sondern auch erstmals die bisher nicht modellierte Buskommunikation als wesentlicher Bestandteil Entfernter Redundanz mitsimuliert werden.

5.2.3.2 Modell

Das im Rahmen dieser Arbeit erstellte Simulink-Modell der elektronisch geregelten Lenkung mit Entfernter Redundanz basiert auf Teilen eines am Lehrstuhl „Verlässlichkeit von Rechensystemen“ zuvor durchgeführten Projektes (Echtle & Tappe, 2006). Während das dort erstellte Modell sich jedoch mit dem FlexRay-Datenbussystem befasste, liegt das Augenmerk

[13] Das Protokoll FlexRay (Jochim, 2007) wurde für den Automobilbereich entworfen. Es beinhaltet bereits Fehlertoleranzmerkmale, wie die Verwendung von zwei redundanten Übertragungskanälen. Die Datenübertragung ist aufgeteilt in ein dynamisches Segment und ein statisches Segment. Durch den zyklischen Kommunikationsschedule kann für Nachrichten im statischen Segment eine maximale Übertragungszeit garantiert werden. Das dynamische Segment ermöglicht ergänzend die bedarfsweise Übertragung unregelmäßig anfallender Daten.

des nun vorliegenden Modells auf den hier neu eingeführten Merkmalen Entfernter Redundanz. Nachfolgende Abbildung 54 zeigt zunächst eine Übersicht über das Gesamtsystem.

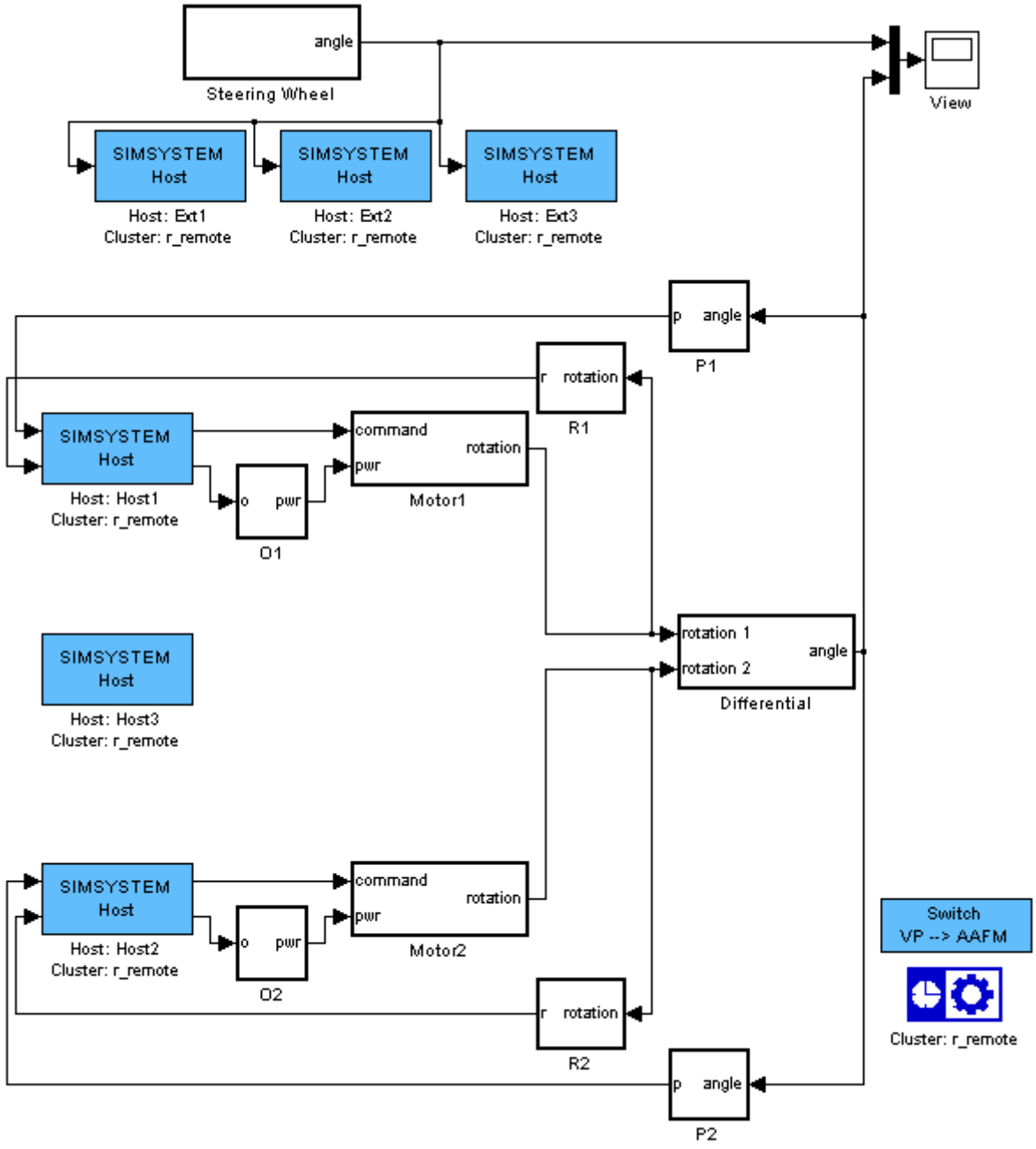


Abbildung 54: Simulationsmodell der elektronisch geregelten Lenkung

Die vom Block Steering Wheel ausgehende Sollvorgabe (d. h. der am Lenkrad eingeschlagene Winkel) wird dreifach redundant abgegriffen (Komponenten Ext1, Ext2 und Ext3) und zunächst über das Bussystem an die drei bereits aus vorangegangenen Modellen bekannten ECUs (hier: Host1, Host2 und Host3) gesendet. In obiger Abbildung ist dieser Vorgang nicht unmittelbar sichtbar, da das Bussystem selbst vom Simulationstool nicht dar-

gestellt wird. Der Lenkwinkel wird ferner an einen Ausgabeblock „View“ übermittelt, welcher es bei Ausführung der Simulation ermöglicht, Sollvorgabe und Reaktion der Lenkung gemeinsam im Zeitverlauf darzustellen.

Die drei Steuergeräte empfangen wie oben beschrieben die Sollvorgabe über das Bussystem. ECU1 und ECU2 steuern jeweils den an sie angeschlossenen Stellmotor an und versorgen ihn über die Endstufen O1 bzw. O2 mit Strom. Für die Überwachung der beiden Motoren sind die Steuergeräte mit Rotationssensoren (R1 bzw. R2), zum Schließen des Regelkreises mit Positionssensoren (P1 bzw. P2) verbunden. ECU3 ist gemäß dem Grundgedanken der Entfernten Redundanz nur mit dem Bussystem verbunden und empfängt alle Sensorwerte mittels Weiterleitung durch ECU1 bzw. ECU2.

Sobald die Motoren Motor1 und Motor2 in Bewegung versetzt werden, wird die resultierende Rotation über ein Differential gekoppelt und in den veränderten Winkel zwischen Radebene und Geradeauslaufstellung umgesetzt. Der so entstandene Lenkwinkel wird von den beiden Positionssensoren P1 und P2 abgegriffen und an den Ausgabeblock geleitet, um ihn mit der vom Lenkrad ausgehenden Sollvorgabe vergleichen zu können.

Beginnend mit dem simulierten Lenkrad (dargestellt in nachfolgender Abbildung 55, in Abbildung 54 als „Steering Wheel“ bezeichnet) wird im Folgenden die Funktionsweise der einzelnen Blöcke des in der Arbeit verwendeten Modells im Detail erläutert.

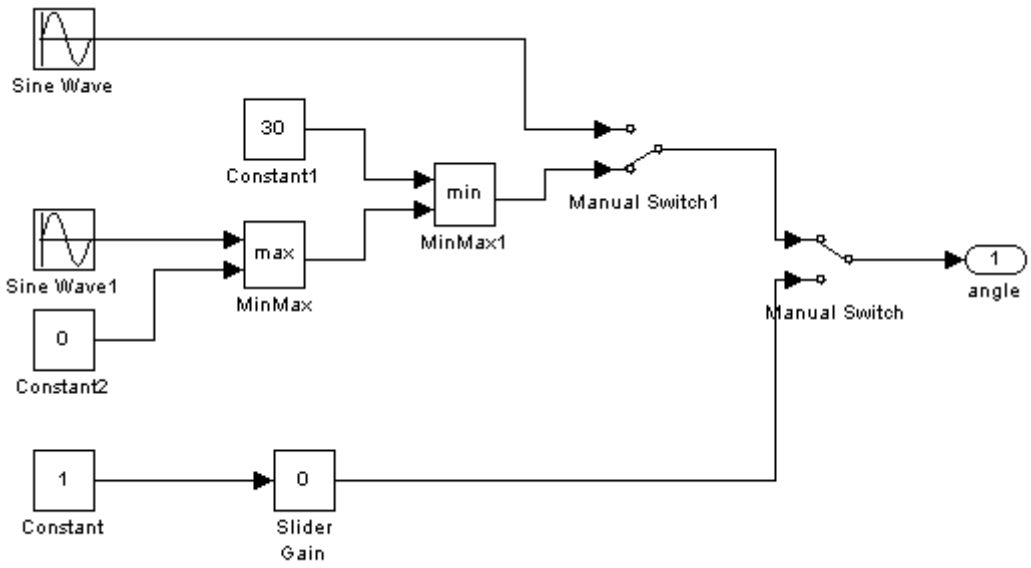


Abbildung 55: Lenkrad (wählbare Szenarien)

Zum Erzeugen von Lenkvorgaben werden drei Alternativen angeboten: eine vollständige Sinuskurve („Schlangenlinien“), eine nach oben durch 30 und nach unten durch 0 begrenzte Sinuskurve („Fahrspurwechsel“) und ein manuelles Lenkrad, welches über einen Schieberegler (s. Abbildung 56) interaktiv beliebige Lenkvorgaben von -50° bis +50° ermöglicht.

Abbildung 56: Lenkrad (Schieberegler)

Das dreifach redundante Senden der Sollvorgabe (Blöcke `Ext1`, `Ext2` und `Ext3` in Abbildung 54) geschieht zeitgesteuert. Abbildung 57 zeigt die Funktionalität eines dieser Blöcke: Der anliegende Sollwert „`angle`“ wird in einer sich alle 2000 µs periodisch wiederholenden Schleife jeweils mit einem Offset von 0 µs vom Block `send_vr` verarbeitet (linker Teil der Abbildung). Innerhalb dieses Blocks (rechter Teil der Abbildung) wird der entsprechende Wert als Nachricht auf das Bussystem gesendet.

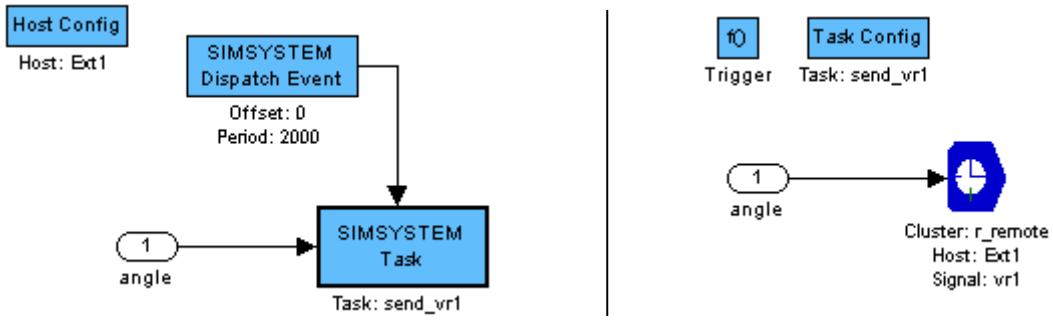


Abbildung 57: Senden der Sollvorgabe

Wie in den bereits beschriebenen Modellen wird die Lenkfunktion selbst von insgesamt drei Steuergeräten (ECUs, in Abbildung 54 als „Hosts“ bezeichnet) erbracht, wobei `ECU1` und `ECU2` jeweils einen Stellmotor ansteuern, während `ECU3` ausschließlich zu Passivierungszwecken dient. Zunächst wird nun die (vom Prinzip her identische) Funktion der beiden erstgenannten ECUs erläutert.

ECU1 und ECU2 sind gleichartig aufgebaut und bestehen beide aus jeweils vier unterschiedlichen sog. „Tasks“, welche nacheinander ausgeführt werden. Am Beispiel von ECU1 werden in Abbildung 58 diese Schritte in ihrer Abfolge dargestellt. In Schritt 1 „send_state“ werden die aktuellen Werte des an das jeweilige Steuergerät angeschlossenen Positionssensors und des entsprechenden Rotationssensors an beide fremden Knoten weitergeleitet. Im anschließenden Schritt 2 „do_control“ wird die eigentliche Ansteuerung vorgenommen, Resultat ist der berechnete Stellwert. Schritt 3 „comp_pass“ betrifft die Passivierung des nicht an diese ECU angeschlossenen Motors (durch Vergleich von dessen Rotation mit dem eigenen Stellwert). Spiegelbildlich werden abschließend in Schritt 4 „rec_pass“ die von beiden fremden Knoten ausgehenden Passivierungsbefehle verarbeitet. Wie aus der Abbildung ersichtlich, dauert auch hier ein Zyklus genau 2000 µs, wobei die unterschiedlichen Aufgaben jeweils mit einem Offset von 0 µs, 500 µs, 800 µs und 1200 µs ausgeführt werden. Diese Latenzen sind notwendig, um die Trägheit der Lenkung sowie die für das Senden von Nachrichten über das Bussystem benötigte Zeit zu berücksichtigen. Für den oben beschriebenen Ablauf erhält das simulierte Steuergerät als Eingangssignale die Werte der an ihn angeschlossenen Sensoren (für ECU1: p1 und r1). Als Ausgangssignal werden nachgelagerten Blöcken der berechnete Steuerbefehl (hier: c1) und die von anderen Knoten empfangenen Passivierungsaufforderungen (hier: o21 und o31) zur Verfügung gestellt.

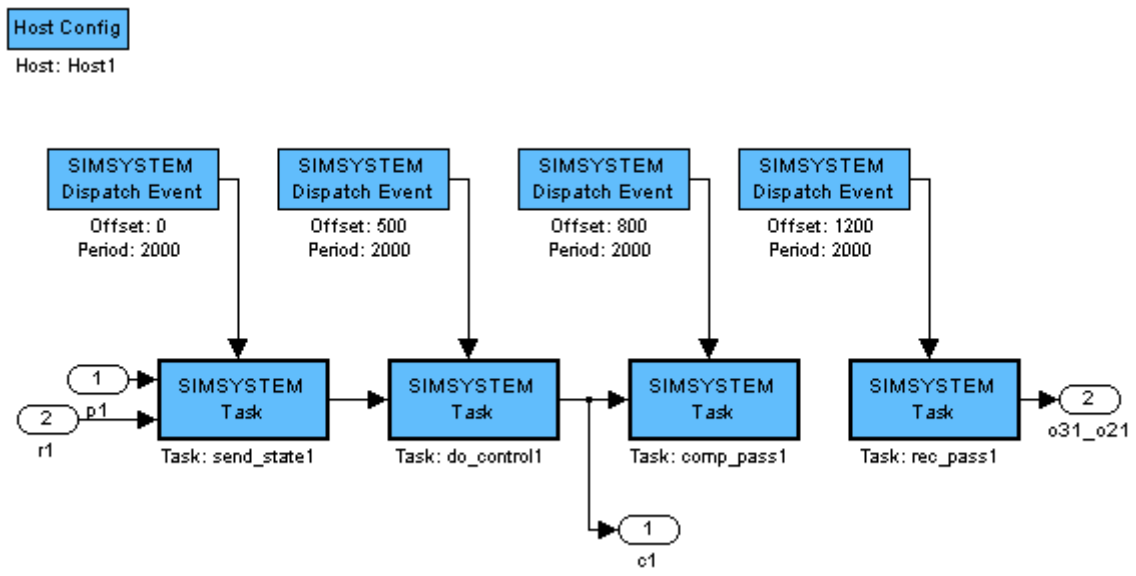


Abbildung 58: Ablauflogik von ECU1

Der erste der oben genannten Schritte, „send_state“, ist in Abbildung 59 dargestellt. Einzige Aufgabe dieses Tasks ist es, die anliegenden Sensorwerte p1 und r1 (Position und Rotation des zu diesem Knoten gehörenden Motors) als Nachricht auf dem FlexRay-Bus anderen Knoten zur Verfügung zu stellen. Der Positionswert wird zudem als Signal p_out an nachfolgende Blöcke innerhalb desselben Knotens durchgeleitet.

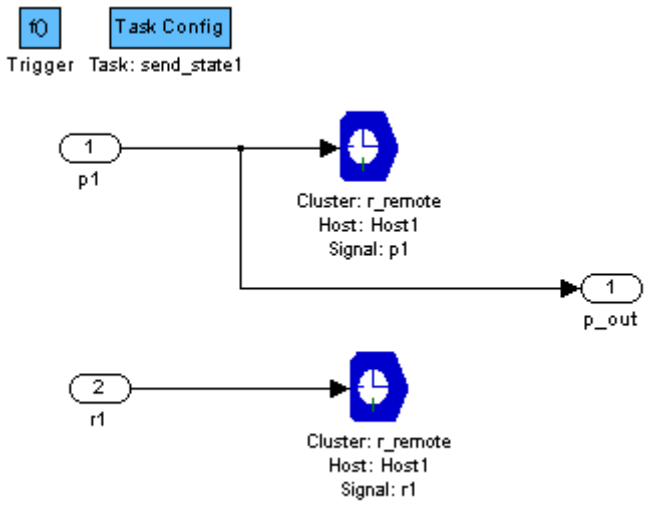


Abbildung 59: Schritt 1, Weiterleiten der Sensorwerte

Im nächsten Schritt, „do_control“ (s. Abbildung 60), findet die Berechnung des Steuerbefehls für den an diese ECU angeschlossenen Motor statt. Dieser ergibt sich wie folgt aus der Differenz von Soll- und Istwert: Der am Lenkrad vorgegebene Sollwert wird zunächst über das Bussystem empfangen. Da der Wert, wie bereits beschrieben, dreifach redundant gemessen und übermittelt wird (vr1, vr2 und v3), ist der Median dieser drei Werte zu berechnen. Dies leistet eine von MATLAB vordefinierte Funktion (Block „MATLAB Fcn“). Als Istwert liegt die vom entsprechenden Sensor gemessene Position p bereits lokal vor. Da diese Information im Sensor mit einer Signatur versehen wird, muss sie an dieser Stelle zunächst dekodiert werden (der Dekodierungsvorgang wird an späterer Stelle separat erläutert). Nachdem nun die Differenz aus Soll- und Istwert berechnet wurde, werden mittels eines weiteren, in Simulink als „Dead Zone“ bezeichneten Blocks sehr kleine Abweichungen ausgefiltert, um ein „Flattern“ der Lenkung zu verhindern. Der sich daraus ergebende Wert wird als Steuerbefehl c an nachfolgende Blöcke durchgeleitet und zu Diagnosezwecken in der sog. MATLAB-Workspace gespeichert.

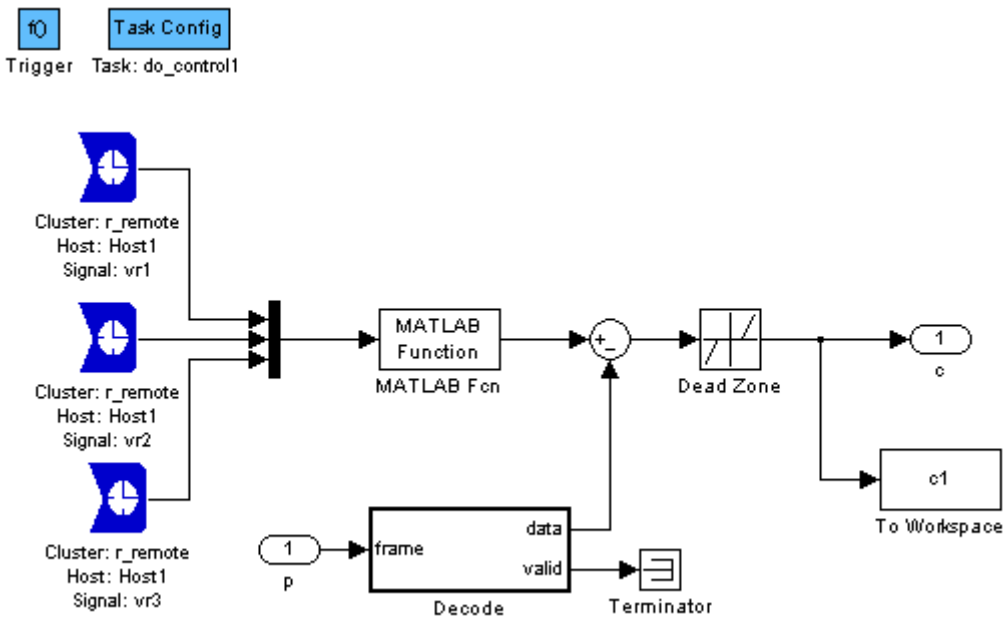


Abbildung 60: Schritt 2, Berechnen des Steuerbefehls

Als dritter Schritt (s. Abbildung 61) findet die Passivierungsentscheidung über den nicht an diesem Knoten angeschlossenen Motor statt. Sie benötigt als Eingänge den über den Bus empfangenen „fremden“ Rotationswert (für die hier beschriebene `ECU1` ist dies also die Rotation von `Motor2`, `r2`) sowie den im vorangegangen Schritt berechneten, eigenen Steuerbefehl `c`. Auch hier ist die über den Bus empfangene Information zunächst zu dekodieren. Ein ungültiger Rotationswert führt dabei unmittelbar zur Passivierung. Die letztendlich getroffene Passivierungsentscheidung wird nun ihrerseits mit einer Sequenznummer und einer Signatur versehen (auch dieser Vorgang wird an späterer Stelle erläutert) und anschließend über den Bus an den ggf. zu passivierenden Knoten gesandt (für `ECU1` also an Knoten 2, da das entsprechende Signal von Knoten 1 ausgeht, wird es als `o12` bezeichnet). Für Diagnosezwecke wird die Information erneut zusätzlich in der MATLAB-Workspace gespeichert.

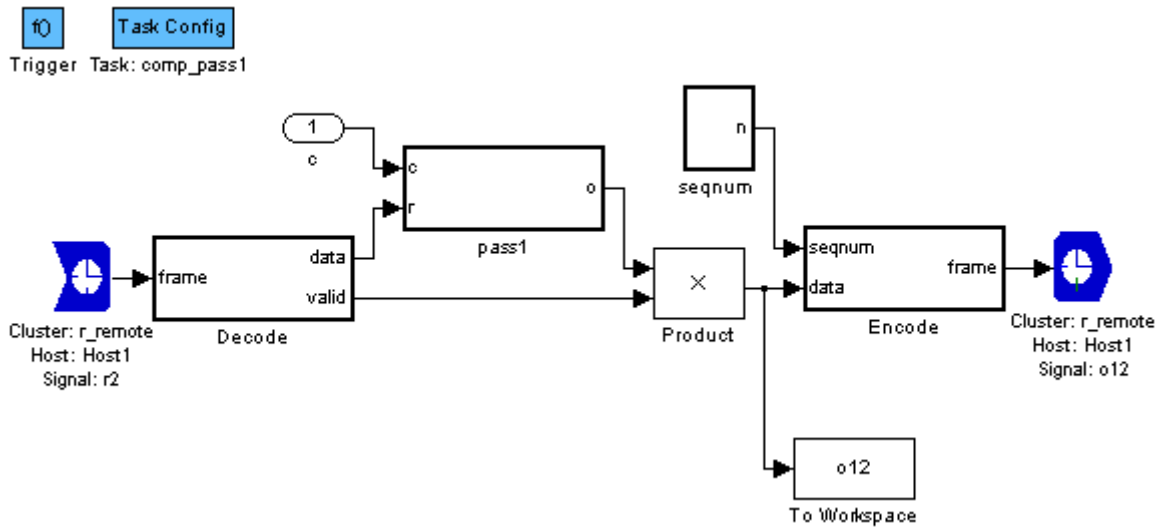


Abbildung 61: Schritt 3, Treffen der Passivierungsentscheidung

Für die in Abbildung 62 dargestellte eigentliche Passivierungsentscheidung werden die letzten drei Steuerbefehle c (erreicht wird dies durch Blöcke vom Typ „Unit Delay“) mit der aktuellen Rotation r verglichen. Wenn diese im Widerspruch zueinander stehen (Block „Fcn“), wird die Passivierungsentscheidung als Signal o nachfolgenden Blöcken bereitgestellt.

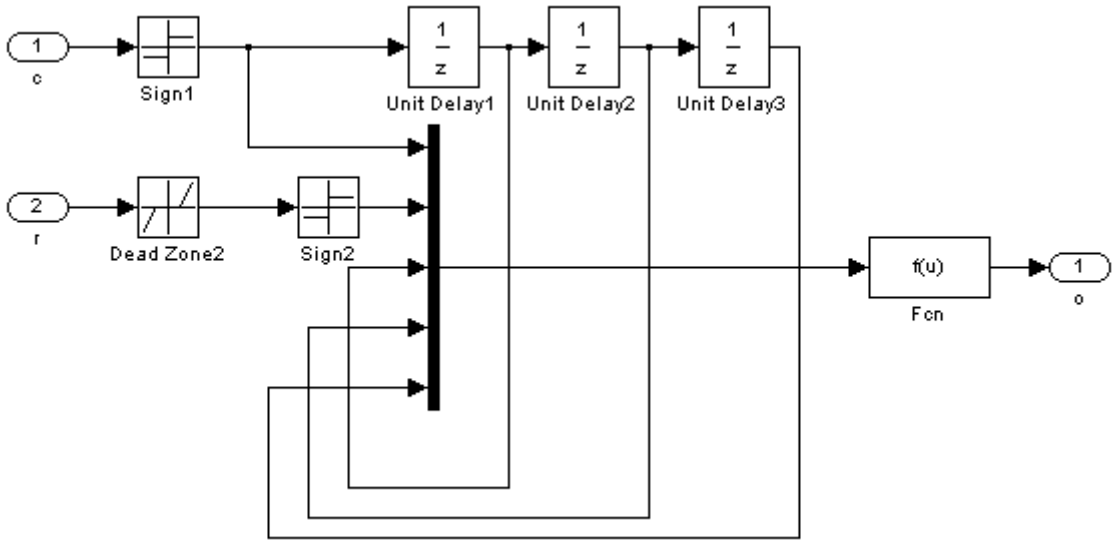


Abbildung 62: Passivierungsentscheidung (Detail)

Als letzter Schritt steht der Empfang der Passivierungssignale fremder Knoten. Dieser Vorgang ist in Abbildung 63 gezeigt. Da hier ECU1 dargestellt ist, wird die Passivierung von Knoten 3 und 2 empfangen (Signale o31 und o21). Diese Werte werden nach dem Empfang als Signal an nachfolgende Blöcke durchgereicht.

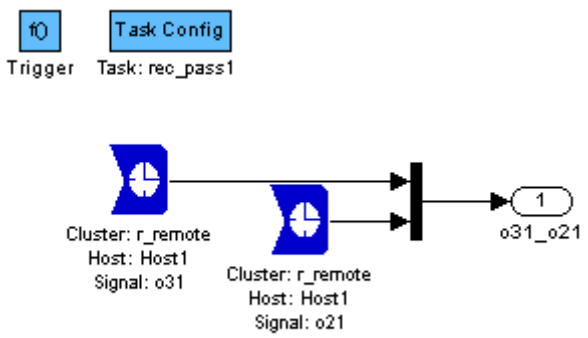


Abbildung 63: Schritt 4, Empfang der Passivierungssignale

Das in Abbildung 58 im Überblick dargestellte Verhalten von ECU1 ist somit nun vollständig beschrieben. Es sei darauf hingewiesen, dass die Abläufe in ECU2 spiegelbildlich, aber ansonsten völlig identisch sind: Knoten 2 sendet die Werte der an ihn angeschlossenen Rotations- und Positionssensoren an ECU1, steuert Motor2 an, trifft anhand des eigenen Stellwerts und der von ECU1 empfangenen Werte eine Passivierungsentscheidung über Motor1 und empfängt die Passivierungskommandos der beiden anderen Knoten für Motor2. Da das vollständige Modell im elektronischen Anhang der Arbeit enthalten ist, werden diese Abläufe hier nicht erneut dargestellt.

Abweichend zu ECU1 und ECU2 stellt sich das Verhalten von ECU3 dar. Dieser Knoten verfügt über keinen eigenen Motor und wird somit lediglich für Passivierungszwecke verwendet. Eine weitere Besonderheit besteht darin, dass der Knoten als Entfernte Redundanz lediglich mit dem Bussystem verbunden ist, aber nicht direkt mit der Peripherie verbunden ist. Der übergeordnete Ablauf der von ECU3 ausgeführten Aufgaben ist in Abbildung 64 dargestellt: Während des 2000 µs dauernden Zyklus wird bei einem Offset von 500 µs bzw. 800 µs jeweils ein Task gestartet. Im ersten Schritt, „do_control“, findet die Berechnung des Steuerbefehls statt; der zweite, „comp_pass“, beinhaltet schließlich das Treffen der Passivierungsentscheidung über die beiden anderen Knoten ECU1 und ECU2.

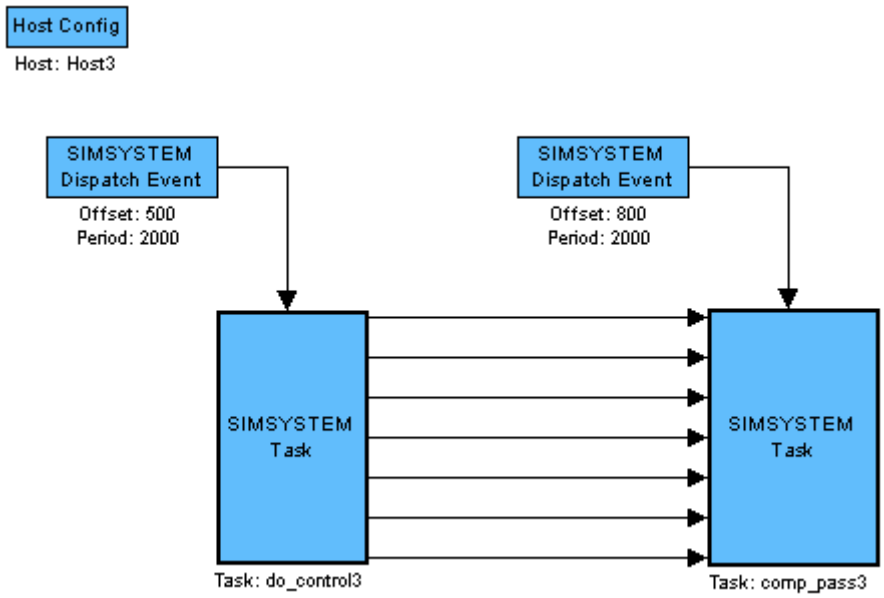


Abbildung 64: Ablauflogik von ECU3

Die Berechnung des Steuerbefehls gestaltet sich in ECU3 aufwändiger als in den anderen beiden ECUs, da – der Idee Entfernter Redundanz entsprechend – keine eigenen Positionssensoren vorhanden sind. Der Ablauf dieses Schrittes ist in Abbildung 65 dargestellt: Auch ECU3 berechnet den Stellwert aus der Differenz zwischen Soll- und Istwert, abzüglich einer Toleranz zum Verhindern eines „Flatterns“ der Lenkung. Der Sollwert wird analog zu ECU1 und ECU2 aus dem Median der drei vom Lenkrad empfangenen Werte vr1, vr2 und vr3 berechnet. Abweichend ist jedoch die Berechnung des Istwertes. Im für die vorliegende Arbeit erstellten Modell werden zunächst die Rotationswerte r1 und r2 beider Motoren verwendet, um einen Näherungswert für den gesuchten Positionswert p3 zu bestimmen (Block calc_p3‘). Anschließend wird aus den empfangenen Positionswerten p1 und p2 derjenige Wert gewählt, der näher an diesem Näherungswert liegt (Block choose_p). Das Ergebnis geht als Positionswert p3 in die Berechnung des von ECU3 verwendeten Steuerbefehls ein. Bevor nun diese Abläufe genauer erläutert werden, sei darauf hingewiesen, dass die Signaturen der empfangenen Sensorwerte (als Signale p1s, p2s, r1s und r2s) an nachfolgende Blöcke weitergeleitet werden und dort ebenfalls in die Passivierungsentscheidung eingehen.

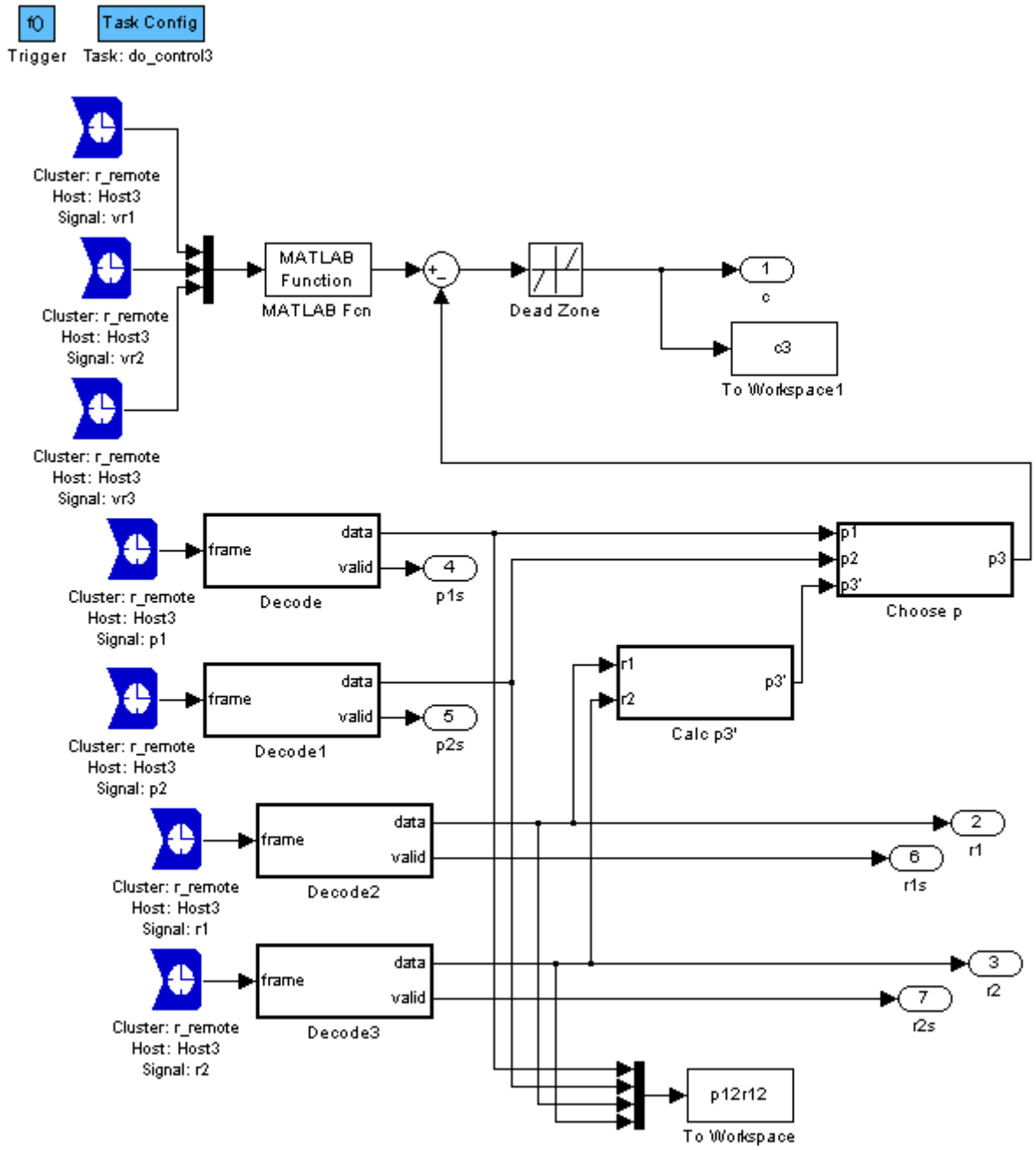


Abbildung 65: Schritt 1, Berechnen des Steuerbefehls

Die Berechnung des Näherungswertes durch den Block calc_p3‘ ist in nachfolgender Abbildung 66 dargestellt. Wie aus der Abbildung ersichtlich, wird mit der Durchschnittsbildung über r1 und r2 zunächst die Funktion eines Differentials nachgebildet. Die sich so ergebende Gesamtrotation wird im Zeitverlauf aufaddiert, wobei ein Korrekturfaktor K verwendet wird, der sich aus der Zykluszeit des Gesamtsystems ergibt (in diesem Fall 2000 µs). Das Ergebnis p3‘ ist ein Näherungswert für die gesuchte Position.

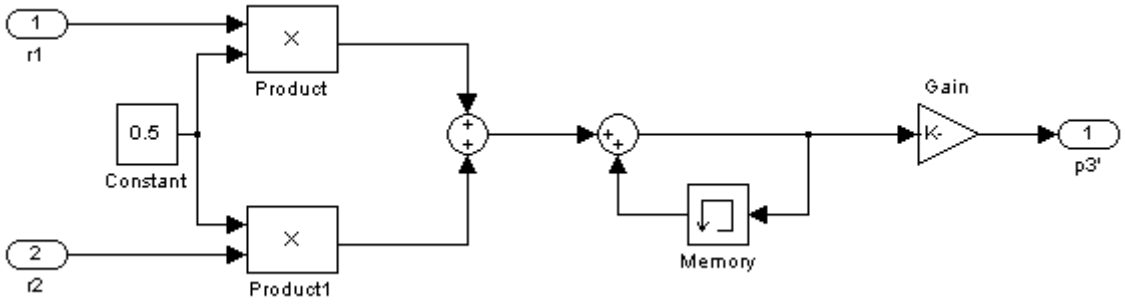


Abbildung 66: Berechnung des Näherungswertes P3‘

Der ermittelte Näherungswert wird nun verwendet, um zu entscheiden, ob p1 oder p2 als Positionswert p3 in die Berechnung des Steuerbefehls eingeht. Dieser Prozess ist in Abbildung 67 dargestellt: Es wird zunächst jeweils der absolute Abstand zwischen p1 und p3‘ sowie zwischen p2 und p3‘ berechnet. Aus beiden Abständen wird sodann die Differenz ermittelt. Ist diese negativ, so ist p1 zu wählen, andernfalls p2.

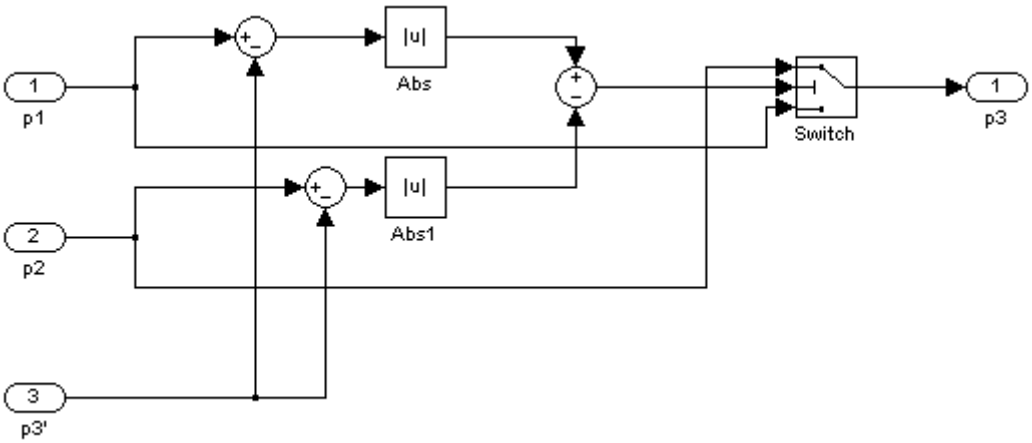


Abbildung 67: Wahl eines Wertes für P3

Der zweite von ECU3 ausgeführte Schritt „comp_pass“ (s. Abbildung 68) beinhaltet das Treffen der Passivierungsentscheidung. Nachdem aus den vorangegangenen Berechnungen ein Steuerbefehl c vorliegt, wird sowohl über Motor 1 als auch über Motor 2 eine Passivierungsentscheidung getroffen (wie bereits in Abbildung 62 dargestellt). Ungültige Signaturen führen dabei ebenfalls zur Passivierung des betreffenden Motors. Der Passivierungsbefehl wird mit einer Sequenznummer versehen, signiert und über den Bus als Signal o31 bzw. o32 an den entsprechenden Knoten gesandt sowie zu Diagnosezwecken auf die MATLAB-Workspace ausgegeben.

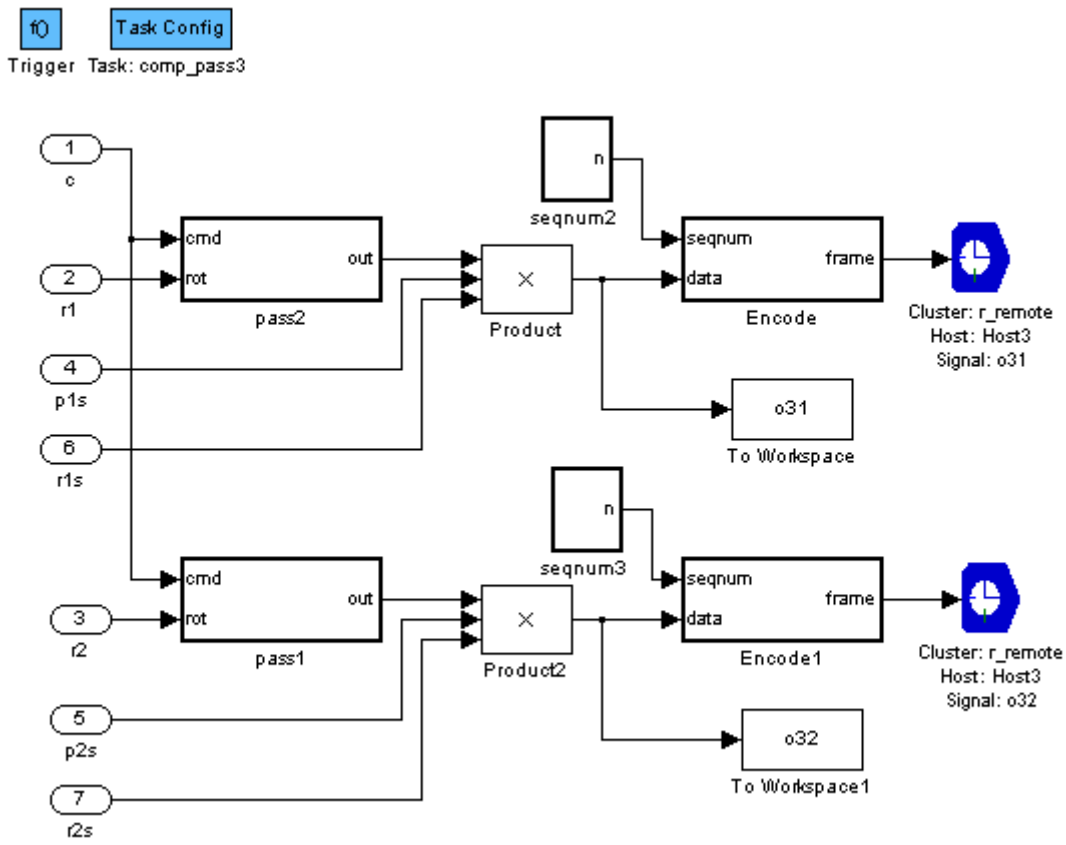


Abbildung 68: Schritt 2, Treffen der Passivierungsentscheidung

Hiermit sind die Funktion von ECU3 und somit auch der gesamte in den Steuergeräten vorliegende Anteil des Systems beschrieben.

Abschließend sei zur Übersicht nochmals die gesamte Kommunikation zwischen allen Steuergeräten dargestellt (s. Tabelle 8): Die drei Steuergeräte Ext1, Ext2 und Ext3 senden jeweils den von ihnen erfassten Sollwert vr1, vr2 bzw. vr3 an alle die Lenkfunktion ausführenden Steuergeräte (Host1, Host2 und Host3). Diese kommunizieren untereinander wie folgt: Host1 sendet die von Sensor R1 gemessene Rotation als r1 an Host2 und Host3. Host3 erhält zusätzlich den Positionswert p1. Umgekehrt sendet Host2 die dort gemessene Rotation r2 zur Kontrolle an Host1 und Host3. Host3 erhält zudem den Positionswert p2. Mit diesen Informationen kann Host1 eine Passivierungsentscheidung über Motor2 treffen (o12) und umgekehrt Host2 über Motor1 (o21). Host3 trifft eine Passivierungsentscheidung über beide Motoren (o31 und o32). Die gesamte hier dargestellte Kommunikation findet, dem Prinzip der Entfernten Redundanz entsprechend, über das Bussystem statt, so dass trotz der gegenseitigen Kontrolle der Knoten keine Verkabelung „über Kreuz“ notwendig ist.

Tabelle 8: Kommunikationsmatrix

von \ *an*	**Host1**	**Host2**	**Host3**
Ext1	vr1	vr1	vr1
Ext2	vr2	vr2	vr2
Ext3	vr3	vr3	vr3
Host1	-	r1, o12	p1, r1
Host2	r2, o21	-	p2, r2
Host3	o31	o32	-

In den nächsten Abschnitten folgt nun die Beschreibung aller weiteren Komponenten des Systems. Es sind dies, wie in Abbildung 54 ersichtlich, die beiden Endstufen O1 und O2, die Motoren Motor1 und Motor2, das simulierte Differential sowie die Sensoren P1, P2, R1 und R2.

Die Endstufen (s. Abbildung 69) entscheiden anhand der beiden von den jeweils anderen Knoten empfangenen Aktivierungs- bzw. Passivierungsnachrichten darüber, ob der an sie angeschlossene Motor mit Strom zu versorgen ist. Die über das Bussystem erhaltenen Nachrichten sind zunächst zu dekodieren. Der Block „decode“ (beschrieben weiter unten in Abbildung 75) liefert die entsprechenden Daten (0 = passivieren, 1 = aktivieren) und die Information, ob die Daten gültig signiert und mit korrekter Sequenznummer versehen wurden (0 = ungültig, 1 = gültig). Damit der Motor mit Strom versorgt wird, muss mindestens eine gültig signierte Aktivierungsnachricht vorliegen (vgl. Kapitel 4.3.3). Der Block „debounce“ (ohne Abbildung) stellt dabei sicher, dass eine zu einem früheren Zeitpunkt bereits umgesetzte Passivierung bestehen bleibt.

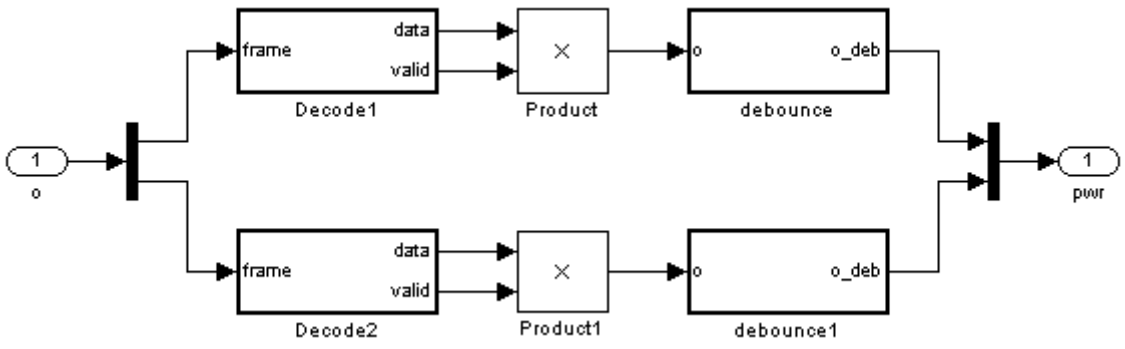


Abbildung 69: Endstufe

Die beiden Motoren (vgl. Abbildung 70) erhalten den Stellbefehl command (– 1, 0 oder 1) der sie ansteuernden ECU sowie die Spannungsversorgung pwr durch die zugehörige Endstufe. Ferner bildet der Block die Trägheit des Motors ab und liefert nach dem Ausfiltern kleinerer Abweichungen denjenigen Wert, der als Rotation in das System eingeht.

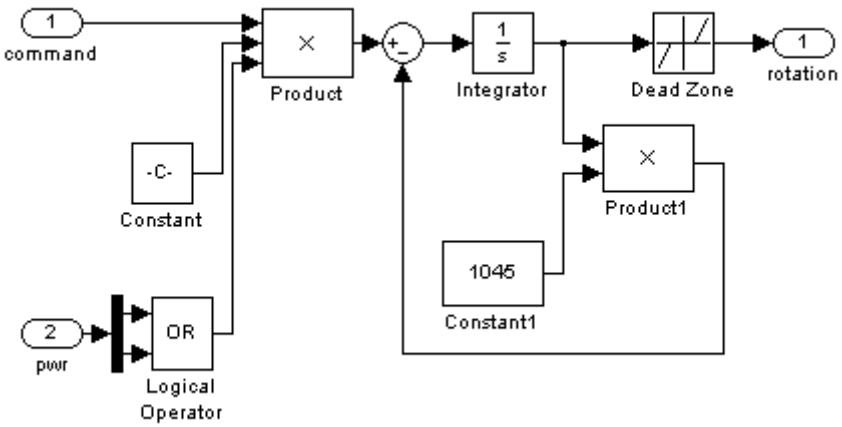


Abbildung 70: Motor

Das in Abbildung 71 dargestellte Differential berechnet zunächst den Mittelwert der (tatsächlichen) Rotationen der beiden Motoren. Das Ergebnis wird über die Zeit integriert, um hieraus den entstehenden Lenkwinkel angle zu berechnen.

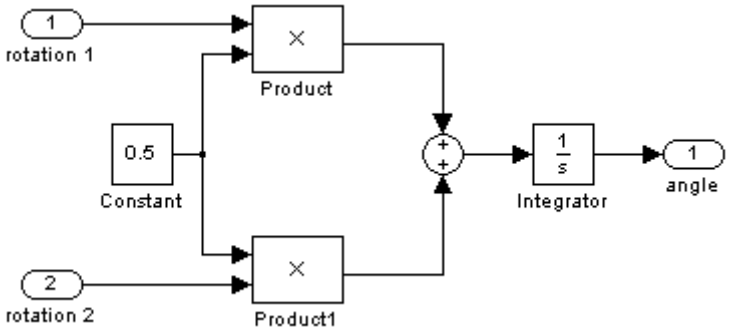


Abbildung 71: Differential

Sensoren (Abbildung 72, hier ein Rotationssensor, für Positionssensoren ist die Darstellung analog) versehen den gemessenen Wert mit einer Sequenznummer und signieren sodann diese Information (der zugehörige Block „encode“ wird in Abbildung 74 erläutert). Die so generierte Nachricht wird vom jeweiligen Steuergerät über das FlexRay-Bussystem versandt.

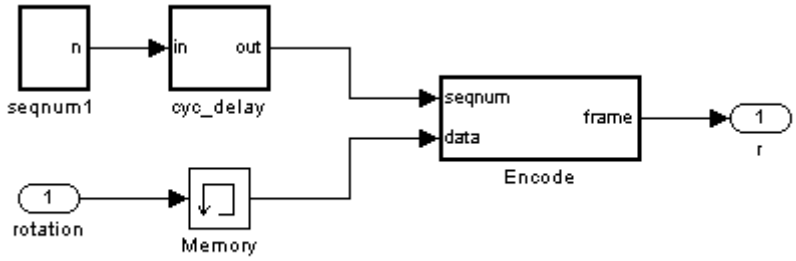


Abbildung 72: Sensor

Das Erzeugen der Sequenznummer (Abbildung 73) besteht dabei aus dem periodischen Hochzählen eines 8-Bit-Zählers.

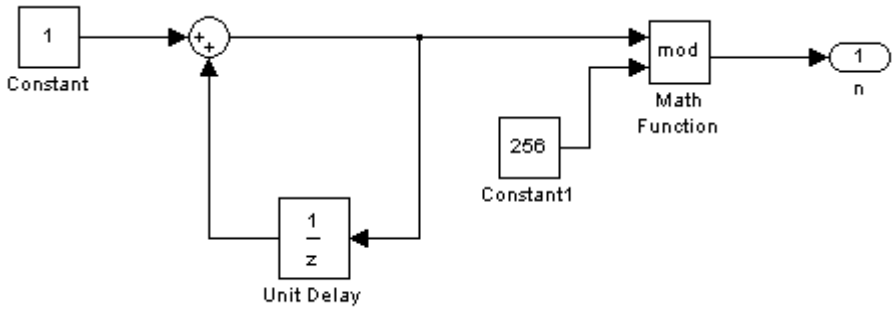


Abbildung 73: Erzeugung von Sequenznummern

Dem Prinzip der Entfernten Redundanz entsprechend sind Informationen, welche durch fremde Knoten weitergeleitet werden, so zu schützen, dass Veränderungen mit hoher Wahrscheinlichkeit zu erkennen sind. Das in Kapitel 4.2 vorgestellte Verfahren wird auch im vorliegenden Modell angewendet (s. Abbildung 74): Zunächst werden Sequenznummer und Nutzdaten aneinandergehängt (Block „conc_n_d“). Anschließend wird über diese Daten ein CRC berechnet (Block „gen_crc“) und mit dem geheimen Faktor signiert (Block „sigma“). Signatur, Sequenznummer und Nutzdaten werden wiederum angehängt (Block „app_sig“) und bilden gemeinsam die letztendlich über den Bus gesendete Nachricht.

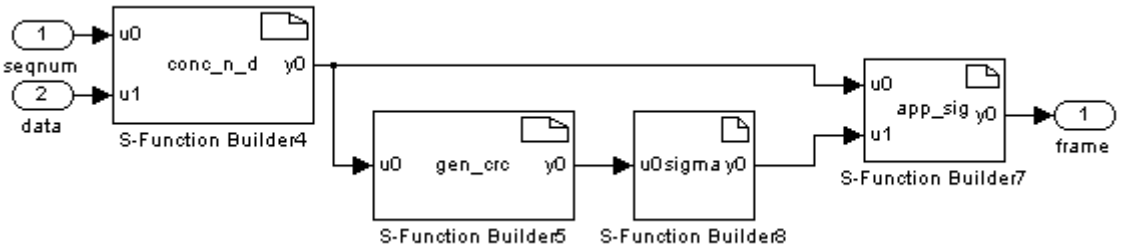


Abbildung 74: Kodieren von Nachrichten

Empfängerseitig ist der oben beschriebene Ablauf in umgekehrter Reihenfolge zu vollziehen: Das Dekodieren einer Nachricht (Abbildung 75) beinhaltet somit zunächst die Zerlegung in ihre Bestandteile: Sequenznummer, Nutzdaten und Signatur. Anschließend werden die Sequenznummer (Block „chk_seqnr_ext“) sowie die Gültigkeit der Signatur (Block „tau“) geprüft. Hierzu ist ebenso erneut der CRC zu berechnen. Eine Nachricht ist genau dann gültig, wenn beide Tests bestanden wurden.

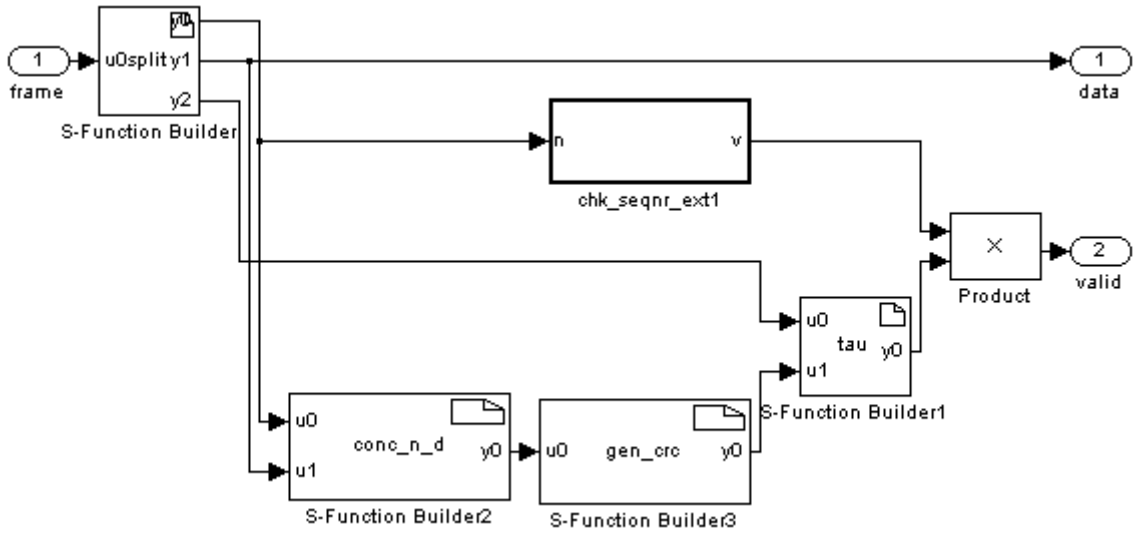


Abbildung 75: Dekodieren von Nachrichten

Bei der Prüfung der Sequenznummer (Abbildung 76) muss unterschieden werden zwischen dem Übergang von einer Zahl zwischen 0 und 254 zu ihrem Nachfolger und dem Sprung zwischen dem größtmöglichen Wert 255 und 0. Diese Unterscheidung findet innerhalb des Blocks „Fcn“ statt. In der Simulation ist ebenso zu berücksichtigen, dass bedingt durch die unterschiedliche Auswertungsreihenfolge der einzelnen Blöcke für eine kurze Zeitspanne ggf. noch kein Nachfolger vorliegt. Im vorliegenden Modell wird diese Prüfung über einen Zähler realisiert.

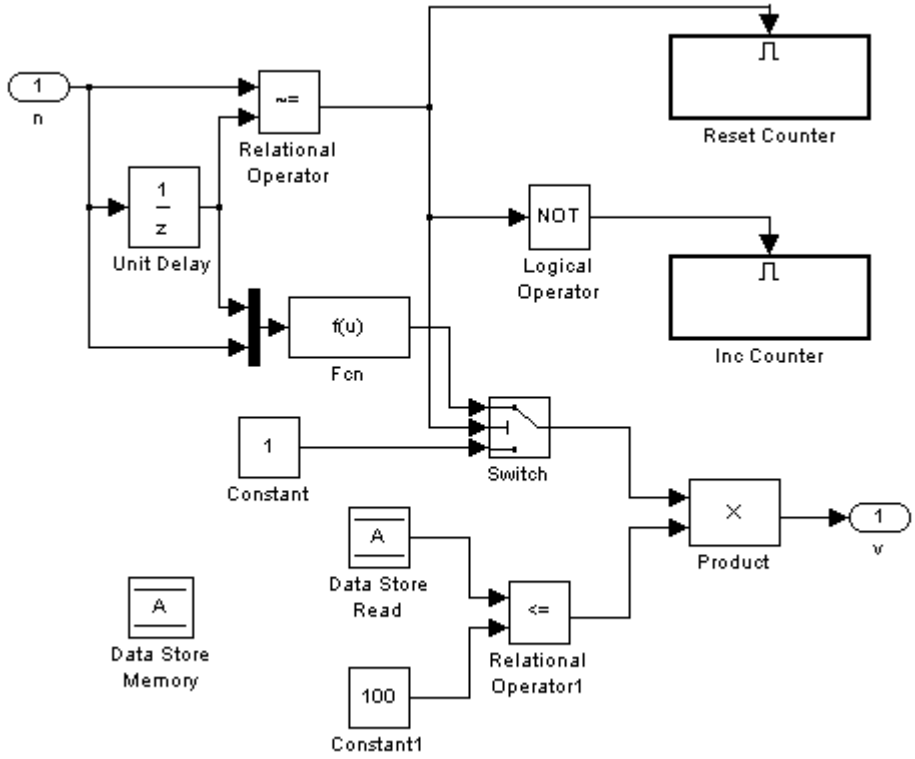


Abbildung 76: Prüfung von Sequenznummern

Die für Entfernte Redundanz erforderlichen Funktionen zur CRC-Berechnung und Signaturerzeugung bzw. -prüfung wurden für das vorliegende Modell in der Programmiersprache C realisiert und mittels Blöcken vom Typ „S-Function Builder“ (vgl. Kapitel 5.2.3.1) in die Simulation integriert.

Abbildung 77 zeigt den Quelltext der CRC-Berechnung. Verwendet wurde das Generatorpolynom $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$ (hexadezimal `0x180F`), welches zunächst um sieben Bit logisch nach links zu verschieben ist, da die zu schützenden Nutzdaten `u0[0]` eine Länge von 20 Bit aufweisen. Sodann wird der Divisionsrest mit der zu sendenden Nachricht initialisiert und die Polynomdivision durchgeführt. Nach Abschluss der Division wird der entstandene Rest (nun wieder entsprechend um acht Stellen nach rechts verschoben) am Ausgang des Blocks als Signal `y0[0]` bereitgestellt.

```
unsigned int message = u0[0];
unsigned int POLY20 = 0x180F << 7; // CRC-12-polynom

unsigned int remainder = message;

unsigned char bit;
for (bit = 20; bit > 0; --bit) {
  if (remainder & 0x80000) {
      remainder ^= POLY20;
  }
  remainder <<= 1;
}

y0[0] = (remainder >> 8);
```

Abbildung 77: CRC-Berechnung

Die Berechnung der Signatur erfolgt im hier beschriebenen Modell nach dem in Kapitel 4.2 vorgestellten Verfahren, der zugehörige C-Quelltext ist in Abbildung 78 dargestellt. Als Modul wird der Wert $m = 2^{12}$ verwendet, das zu signierende Eingangssignal `u0[0]` ist der nach obiger Vorschrift berechnete CRC einer Nachricht. Die hierüber gebildete Signatur `s` wird als Ausgangssignal `y0[0]` nachfolgenden Blöcken zur Verfügung gestellt.

```
#define m 0x1000                        // module m = 4096 (2^12)
unsigned int crc = u0[0];               // message-crc
unsigned int s = (crc * (int)*a) % m;

y0[0] = s;
```

Abbildung 78: Bilden der Signatur

Auch die Prüfung der Signatur erfolgt nach dem aus Kapitel 4.2 bekannten Verfahren. Abbildung 79 zeigt den Quelltext des entsprechenden C-Programms. Erneut wird der Wert $m = 2^{12}$ als Modul verwendet. Verglichen werden stets die empfangene Signatur im Eingangssignal `u0[0]` und der über die Nutzdaten berechnete CRC im Signal `u1[0]`. Das Ergebnis der Prüfung `t` wird als Signal `y0[0]` am Ausgang des Blocks bereitgestellt.

```
#define m 0x1000            // module m = 4096 (2^12)
unsigned int s = u0[0];     // received signature
unsigned int crc = u1[0];   // calculated crc

unsigned int t = (s * (int)*b) % m == (crc * (int)*c) % m;

y0[0] = t;
```

Abbildung 79: Prüfen der Signatur

Die für die Signaturerzeugung und -prüfung schließlich benötigten Faktoren wurden zufällig durch Festlegung der einzelnen Bits per Münzwurf gewählt (allerdings mit maximal einer führenden Null), wobei sich der dritte Faktor gemäß Kapitel 4.2 jeweils durch Multiplikation modulo m aus den ersten beiden ergibt. Tabelle 9 enthält die für die verschiedenen Nachrichten des Lenkungsmodells verwendeten Werte.

Tabelle 9: Zufällig gewählte Schlüssel

Nachricht	geheim	öffentlich	
o12	0xACB	0xCD2	0xE86
o21	0x7C6	0xDBF	0xABA
o31, o32	0x4C9	0xCDA	0xF2A
r1	0x69B	0x989	0xBF3
r2	0xC1D	0x465	0xB71
p1	0xE5C	0x7B7	0x7C4
p2	0x6B3	0x93E	0xA5A

Zum Einsatz kamen noch weitere C-Programme, welche vornehmlich der leichteren Verarbeitung von Nachrichten dienen (Zerlegen in einzelne Bestandteile, Zusammenfügen dieser Bestandteile etc.). Diese Funktionen sind hier nicht abgebildet, befinden sich jedoch mit dem gesamten Modell im elektronischen Anhang der Arbeit.

Auch das vorliegende Simulationsmodell wurde zur Untersuchung der Fehlertoleranzeigenschaften über die eigentliche Systemfunktion hinaus um eine Komponente erweitert, welche als Fehlerinjektor die gezielte Verfälschung einzelner Nachrichten ermöglicht (s. Abbildung 80).

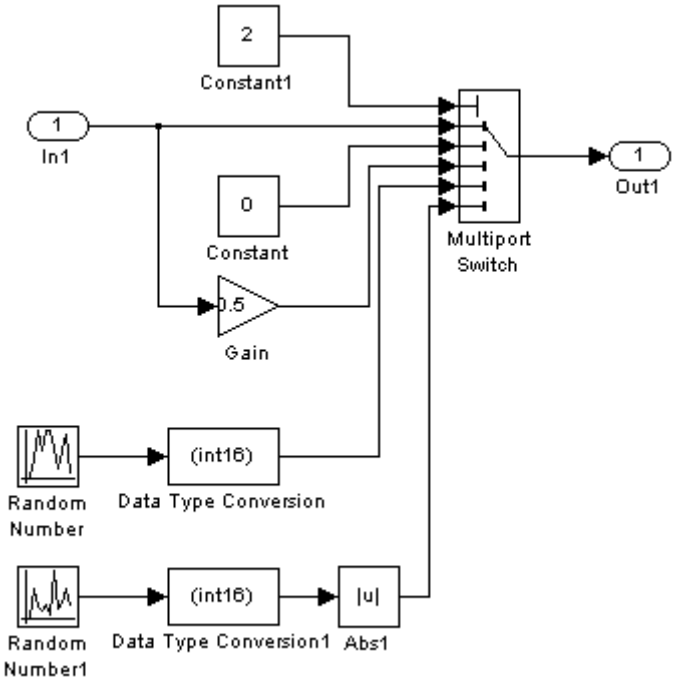


Abbildung 80: Block zur Fehlerinjektion

Wie aus der Abbildung ersichtlich, lassen sich jeweils manuell die Fehlerarten „konstanter Wert“, „Multiplikation mit einem beliebigen Faktor“, „Zufallszahl“ und „positive Zufallszahl“ wählen. Durch Einfügen des entsprechenden Blocks in die verschiedenen Komponenten des Modells konnten so die Fehlertoleranzeigenschaften bei Vorliegen bestimmter Fehler überprüft werden.

5.2.3.3 Ergebnis

Das im vorangegangenen Abschnitt beschriebene Simulationsmodell wurde für die vorliegende Arbeit nach seiner Erstellung umfassenden Fehlerinjektionsexperimenten unterworfen. Als Szenario wurde dabei die Eingabevariante „Spurwechsel“ (in der Regelungstechnik entsprechend einem „Doppelsprung“) gewählt. Der Lenkwinkel wird hierbei unmittelbar nach Beginn der Simulation von 0° auf 30° und anschließend wieder zurück auf 0° gesetzt. Abbildung 81 zeigt die erwartete Ausgabe bei diesem Szenario: vom Simulationstool gezeichnet wird als gelbe Linie die Sollvorgabe des Fahrers und als magentafarbene Linie die Reaktion der simulierten Lenkung. Die Lenkung folgt dabei während des gesamten Verlaufs mit einer geringen zeitlichen Verzögerung der Sollvorgabe, erreicht jedoch erwartungsgemäß aufgrund der Trägheit der Motoren sowie der im Modell berücksichtigten Toleranz nie exakt denselben Wert.

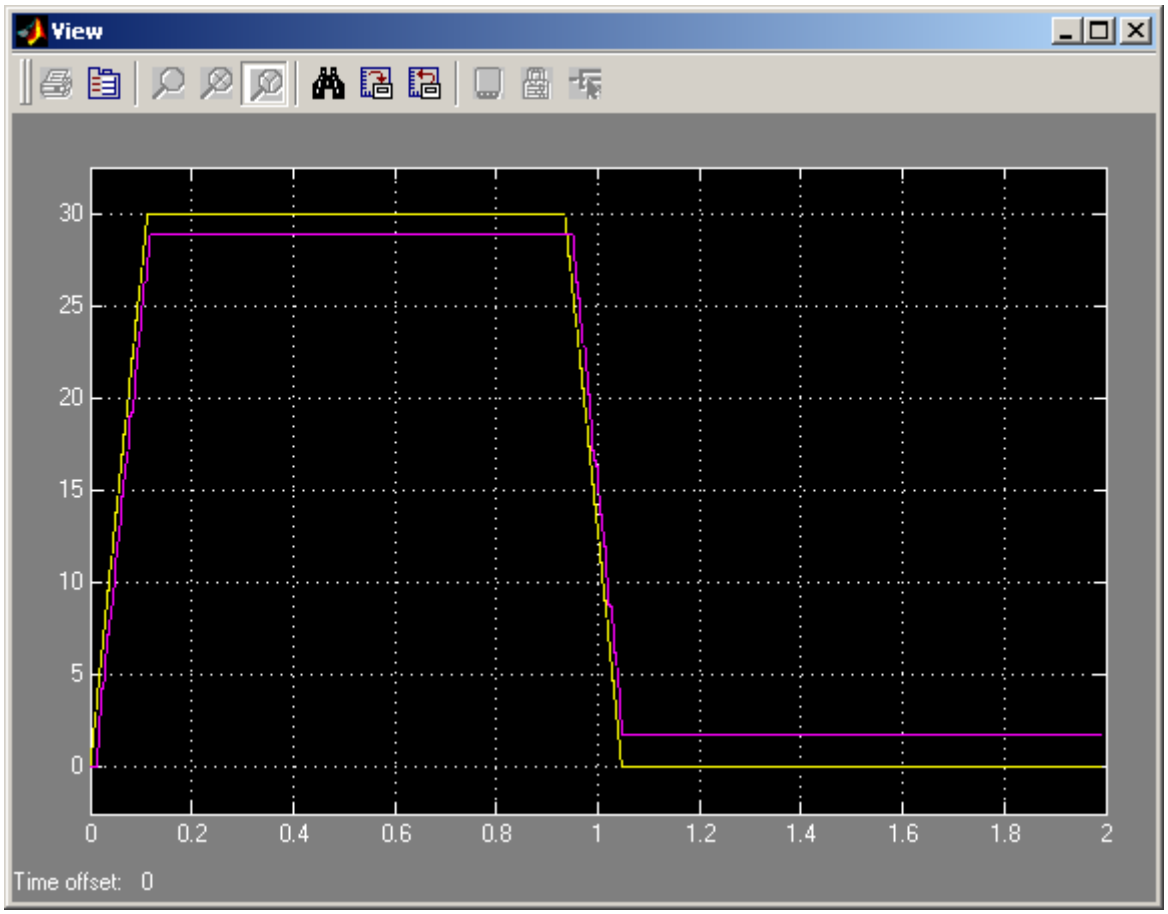


Abbildung 81: Ausgabe der Simulation bei Szenario „Spurwechsel“

Geprüft wurde nun mittels einer entsprechenden Zusicherung im Modell, ob die Differenz zwischen Sollvorgabe und Istwert auch bei Injektion der zu tolerierenden Fehler des Fehlermodells aus Tabelle 1 stets innerhalb einer Spanne von maximal 4° bleibt (dieser Wert ergibt sich anwendungsabhängig aus der Geschwindigkeit der simulierten Motoren und der verwendeten Schalthysterese unter der Annahme, dass nur noch ein Motor seine Arbeit verrichten kann).

Die Ergebnisse dieser Experimente sind in nachfolgender Tabelle 10 aufgeführt. Im elektronischen Anhang der Arbeit befinden sich zusätzlich die Ausgabedateien mit den vom Simulationstool im Zeitverlauf protokollierten Größen. Es sind dies die Steuerbefehle aller drei ECUs, die von den Sensoren gelieferten Positions- und Rotationswerte sowie der Aktivierungs- bzw. Passivierungsstatus der Endstufen.

Tabelle 10: Ergebnisse der Fehlerinjektionsexperimente

Fehlerart	Fehlerort	Zusicherung	Erläuterung der Ausgabe
(kein Fehler)	-	✓	alle Steuerbefehle sowie Position und Rotation einheitlich, keine Passivierung
Motor1/2: Halt bzw. halbe Geschwindigkeit	Ausgang Motor1/2	✓	halbe bzw. keine Umdrehung des betroffenen Motors, bei Halt erfolgt Passivierung
R1/2: zufälliger Wert aus [-800, ..., 800]	Eingang R1/2	✓	betroffener Motor wird von den beiden fremden ECUs passiviert
R1/2: Signatur ungültig	ECU2/1: comp_pass ECU3: do_control	✓	betroffener Motor wird von den beiden fremden ECUs passiviert
P1/2: zufälliger Wert aus [-30, ..., 30]	Eingang P1/2	✓	betroffener Motor wird von beiden fremden ECUs passiviert, betroffene ECU versucht ggf., den jeweils anderen Motor ebenfalls zu passivieren, ist aber in der Unterzahl, da sich der von ECU3 berechnete Näherungswert für P3 beim fehlerfreien Wert P2/1 stabilisiert
P1/2: Signatur ungültig	ECU3: do_control	✓	betroffene ECU versucht ggf., den fremden Motor zu passivieren, ist aber in der Unterzahl
ECU1/2/3: zufälliger Steuerbefehl aus [-1, 0, 1]	ECU1/2/3: do_control	✓	ECU1/2: betroffener Motor wird passiviert, betroffene ECU versucht ggf., den fremden Motor ebenfalls zu passivieren, ist aber in der Unterzahl ECU3: versucht beide Motoren zu passivieren, hierfür fehlt aber die Stimme von ECU1 bzw. ECU2
O1/2: zuf. Wechsel zw. „an“ und “aus“	Ausgang O1/2	✓	zwischenzeitliche Abschaltung des betroffenen Motors, jedoch keine Passivierung durch ECUs

Zusammengefasst bestätigen die oben beschriebenen Ergebnisse die Resultate der bereits durchgeführten Untersuchungen, nach denen Einfehlertoleranz vorliegt und Entfernte Redundanz somit bei geringeren Kosten dieselben Fehlertoleranzeigenschaften wie Dedizierte Redundanz erzielen kann. Zusätzlichen Wert erhält das Simulationsmodell jedoch dadurch, dass das Beispielsystem der elektronisch geregelten Lenkung erstmals relativ detailliert modelliert werden konnte:

- Die mechanischen Komponenten des Lenkungssystems (Motoren, Differential) konnten in realistischer Form nachgebildet werden.
- Die durch Verwendung Entfernter Redundanz zusätzlich erforderliche Buskommunikation wurde unter Verwendung eines virtuellen FlexRay-Datenbussystems komplett mitsimuliert.
- Das in Kapitel 4.2 vorgestellte Schema zur Signaturerzeugung und -prüfung konnte im Rahmen des Modells vollständig umgesetzt werden.

Insofern stellt das vorliegende Simulationsmodell durch seine Detailtreue auch einen Machbarkeitsnachweis dar, insbesondere was das geforderte Echtzeitverhalten und die durch Entfernte Redundanz zusätzlich erforderliche Buskommunikation betrifft. Dieser Nachweis ist allerdings insofern (noch) zu relativieren, als es sich um eine reine Simulation unter weitgehend idealisierten Rahmenbedingungen handelt.

5.2.4 Prototypische Implementierung

5.2.4.1 Methode

Nachdem im Rahmen der im vergangenen Abschnitt beschriebenen funktionalen Simulation der elektronisch geregelten Lenkung alle Hardwarekomponenten in Software abgebildet wurden, soll diese Abstraktionsebene bei der im Folgenden beschriebenen prototypischen Implementierung für Teile des Systems wegfallen.

Der Fokus liegt dabei auf der Implementierung der für Entfernte Redundanz erforderlichen Signaturerzeugung und -prüfung. Entsprechende Funktionen sind zwar bereits im Simulationsmodell enthalten gewesen, dort jedoch nicht im Detail analysiert worden. Ziel ist es nunmehr, diese Untersuchung nachzuholen und zugleich die Realisierbarkeit der für Entfernte Redundanz benötigten Funktionalität in Hardware zu demonstrieren. Von der Anwendungsfunktion der Lenkung wird dabei abstrahiert, jedoch werden die typischen tech-

nischen Rahmenbedingungen des Automobilbaus als einem für Entferne Redundanz möglichen Einsatzgebiet berücksichtigt.

Große Teile des im Rahmen dieser Arbeit entstandenen Demonstrators sind vollständig durch Verwendung kommerziell verfügbarer Hardware erstellt worden (insbesondere das FlexRay-Bussystem). Andere Teile sind insofern zunächst noch ein Modell, als sie durch programmierbare Hardware umgesetzt wurden. Diese Anteile des Systems stehen im Mittelpunkt der nun folgenden Betrachtungen. Zunächst soll jedoch letztmalig die verwendete Modellwelt beschrieben werden.

Beim Schaltungsentwurf bedient man sich oftmals spezieller Hardwarebeschreibungssprachen. Diese zunächst hardwareunabhängigen Beschreibungen werden schrittweise verfeinert und von sog. EDA[14]-Tools in eine hardwareabhängige Schaltungsbeschreibung („Netzlisten“) überführt, wobei die im Zuge dieser Überführung zunehmende Komplexität dem Anwender weitgehend verborgen bleiben kann.

Als Hardwarebeschreibungssprache wurde im Rahmen dieser Arbeit das im europäischen Raum als Standard geltende VHDL[15] verwendet. VHDL-Modelle erlauben es, Schaltungen in ihrer Struktur und Funktion textbasiert zu beschreiben. Das gesamte Design besteht dabei stets aus einer Menge von Blöcken, welche nebeneinander stehen oder ineinander verschachtelt sein können und über Signale miteinander kommunizieren.

Für jeden Block beschreibt eine so genannte **Entity** die Schnittstelle nach außen, eine oder mehrere **Architectures** die Funktionsweise. Die Unterscheidung zwischen Entity und Architecture liegt darin begründet, dass für die Beschreibung der Funktion drei verschiedene Modellierungsstile existieren, welche sowohl nebeneinander verwendet als auch gemischt werden können (und oft auch werden). Man unterscheidet nach den Kernmerkmalen:

- *Structural Description:* Instanziierung und Verknüpfung bereits bestehender Komponenten
- *Data Flow-Description:* Verwendung von Registern und Schaltnetzen, insgesamt typischerweise als Zustandsmaschine formuliert
- *Behavioral Description:* Definition einer Menge zueinander nebenläufiger Prozesse, welche intern sequenziell ablaufen

[14] Electronic Design Automation
[15] VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

Grundsätzlich können unabhängig vom gewählten Modellierungsstil alle Modelle in eine Schaltungsbeschreibung (typischerweise beschrieben im EDIF[16]-Format) überführt werden. Ausgenommen sind jedoch einzelne Sprachelemente, wie beispielsweise Angaben zu zeitlichen Verzögerungen („Delays“), bestimmte Schleifenkonstrukte und etwa der Zugriff auf Dateien.

Als Beispiel für ein einfaches VHDL-Modell soll der in Abbildung 82 dargestellte Halbaddierer dienen:

(Klenke, 1999)

Abbildung 82: Halbaddierer (Schaubild)

Die im obigen Bild bereits ersichtliche Schnittstelle der Schaltung nach außen wird als sog. Entity nahezu identisch in den VHDL-Code übernommen (s. Abbildung 83). Diese definiert den Namen der Komponente sowie Bezeichnung, Datentyp und Richtung der verwendeten Signale.

```
ENTITY half_adder IS

    PORT( x, y, enable: IN BIT;
          carry, result: OUT BIT);

END half_adder;
```

(Klenke, 1999)

Abbildung 83: Halbaddierer (Entity-Definition)

Das Verhalten der Schaltung, die sog. „Architecture“ (s. Abbildung 84), besteht hier lediglich aus einem einzigen Prozess. Dieser stellt, sofern das Eingangssignal `enable` den Wert `1` hat, an den Ausgängen `result` und `carry` die Summe bzw. den Übertrag der Addition der beiden Größen `x` und `y` bereit. Hat `enable` den Wert `0`, so liegt an beiden Ausgängen der Wert `0` an.

[16] Electronic Design Interchange Format

```
ARCHITECTURE half_adder_a OF half_adder IS
    BEGIN
      PROCESS (x, y, enable)
        BEGIN
                IF enable = '1' THEN
                        result <= x XOR y;
                        carry <= x AND y;
                ELSE
                        carry <= '0';
                        result <= '0';
                END IF;
        END PROCESS;
    END half_adder_a;
```

(Klenke, 1999)

Abbildung 84: Halbaddierer (Architecture)

Die so definierte Schaltung kann nun simuliert oder in eine hardwareabhängige Schaltungsbeschreibung überführt werden. Oftmals finden beide Schritte in Folge statt, wobei die Simulation einer ersten Fehlerkontrolle dient. Die bei der Synthese erzeugte Schaltungsbeschreibung kann sodann verwendet werden, um sie in einem programmierbaren Logikbaustein, also etwa einem FGPA[17], zur Ausführung zu bringen. Dieser Weg wurde auch in der vorliegenden Arbeit gewählt.

5.2.4.2 Modell

Die in diesem Teil der Arbeit erstellen Modelle entstanden in einem dreischrittigen Prozess:

1. Zunächst wurden die für Entfernte Redundanz benötigten Komponenten zur Signaturerzeugung und -prüfung gemäß dem in Kapitel 4.2 vorgestellten Schema in der Sprache VHDL implementiert und mittels Schaltungssimulation untersucht.
2. Anschließend wurde die Schaltung auf einem FPGA zur Ausführung gebracht und mit einer Testumgebung versehen.
3. Schließlich wurde dieses System um eine Datenübertragung über ein FlexRay-Bussystem erweitert. Der so entstandene Demonstrator diente als Machbarkeitsstudie und zur Untersuchung des in Kapitel 4.4 erläuterten Schemas der Sequenznummernzerlegung.

[17] Field Programmable Gate Array

VHDL-Implementierung

Für das im Rahmen dieser Arbeit verwendete Schema werden auch bei einer Hardwareimplementierung die CRC-Berechnung, die Signatur mit einem geheimen Faktor und die Prüfung dieser Signatur als Funktionalität benötigt. Für das im vorangegangenen Abschnitt beschriebene Simulationsmodell wurden diese Komponenten bereits in der Programmiersprache C implementiert. Diese Implementierung wurde nunmehr zur Schaltungsbeschreibung in die Sprache VHDL übertragen. Die Algorithmen selbst konnten dabei in nahezu identischer Form erhalten bleiben.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY sigma IS
  -- Signaturerzeugung über Nutzdaten
  PORT (
    clk: IN std_logic;
    i_data: IN std_logic_vector (20 DOWNTO 1);
    o_frame: OUT std_logic_vector (32 DOWNTO 1)
  );
END sigma;

ARCHITECTURE behavior OF sigma IS
  CONSTANT POLY20: std_logic_vector (20 DOWNTO 1) :=
    "11000000011110000000"; -- 180F&"0000000"
  CONSTANT a: integer := 655;
  CONSTANT m: integer := 4096;

BEGIN
  P: PROCESS(clk)
    VARIABLE remainder: std_logic_vector (20 DOWNTO 1);
    VARIABLE crc_asInt: integer range 0 to m-1;
    VARIABLE sign_asInt: integer range 0 to m-1;
  BEGIN
    IF (clk'event AND clk='1') THEN
      remainder := X"FFF00" XOR i_data;

      FOR I IN 20 DOWNTO 1 LOOP
        IF (remainder(20) = '1') THEN
          remainder := remainder XOR POLY20;
        END IF;
        remainder := remainder(19 DOWNTO 1) & '0';
      END LOOP;

      crc_asInt := to_integer(unsigned(remainder(20 DOWNTO 9)));
      sign_asInt := (crc_asInt * a) mod m;

      o_frame <= i_data & std_logic_vector(to_unsigned(sign_asInt, 12));
    END IF;
  END PROCESS P;
END behavior;
```

Abbildung 85: Signaturerzeugung (VHDL-Code)

In vorstehender Abbildung 85 wird nun zunächst die Signaturerzeugung dargestellt. Die Schaltung nimmt Eingabewörter von 20 Bit Länge entgegen (`i_data`). Diese bestehen in späteren Anwendungen aus einer 8 Bit langen Sequenznummer und 12 Bit Nutzdaten, werden hier jedoch zufällig gewählt. Am Ausgang stellt die Schaltung ein 32 Bit langes Wort (`o_frame`), bestehend aus den unverschlüsselten Eingabedaten und der zugehörigen Signatur bereit. Da es sich um eine getaktete Schaltung handelt, beinhaltet die Schnittstellendefinition ferner ein Clocksignal (`clk`), um die Berechnung jeweils bei einer steigenden Flanke (`clk'event AND clk='1'`) auslösen zu können.

Die Signaturerzeugung selbst ist als VHDL-Prozess implementiert und somit praktisch identisch zur bereits beschriebenen C-Implementierung: Zunächst wird ein CRC über die Eingabedaten berechnet. Der Rest der Polynomdivision wird anschließend mit dem geheimen Faktor `a` signiert. Da als Modul des Signaturschemas der Wert $m = 2^{12}$ verwendet wird, ist die Signatur ebenfalls 12 Bit lang. Ergänzt um einige von VHDL geforderte Typkonversionen bilden diese Berechnungen zusammen mit der abschließenden Konkatenation von Eingabedaten und Signatur die Signaturerzeugung.

Nachfolgende Abbildung 86 beinhaltet den VDHL-Code zur Signaturprüfung. Eingabeworte sind hier die über das Bussystem empfangenen Nutzdaten zusammen mit ihrer Signatur (`i_frame`, Länge: 32 Bit). Das Ausgangssignal (`o_good`) liefert „`1`“, wenn der Signaturtest bestanden wurde, andernfalls „`0`“. Erneut ein Clocksignal (`clk`) verwendet, welches die Reaktion auf eine steigende Taktflanke ermöglicht.

Auch hier ist die Berechnung selbst analog zum C-Programm, ergänzt um einige Typkonversionen. Gemäß dem bekannten Schema wird zunächst der CRC über die empfangenen Daten berechnet. Anschließend wird mittels der öffentlichen Faktoren `b` und `c` die Gültigkeit der empfangenen Signatur geprüft.

Zu Testzwecken wurde die Schaltungsbeschreibung um eine nachgebildete Datenübertragung mit der Möglichkeit zur Fehlerinjektion ergänzt. Da Leitungsfehler durch Rauschen typischerweise als sog. Bündelfehler vorliegen, welche sich in beliebiger Weise auf eine bestimmte maximale Anzahl hintereinanderfolgender Bits auswirken, wurde während einer simulierten Datenübertragung diese Fehlerart eingestreut und die empfängerseitige Erkennungsrate mit der eines herkömmlichen CRCs verglichen. Zum Einsatz kam hierbei das Simulationswerkzeug ModelSim der Firma Mentor Graphics.[18]

[18] http://www.mentor.com/

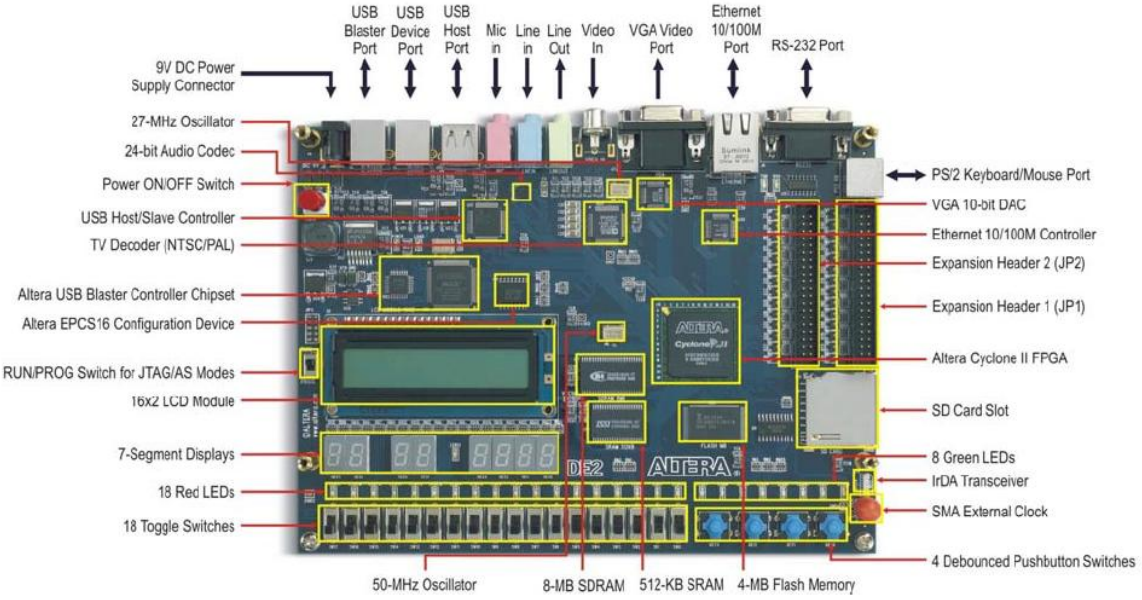
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY tau IS
  -- Prüfung der Signatur eines empfangenen Frames
  PORT (
    clk: IN std_logic;
    i_frame: IN std_logic_vector (32 DOWNTO 1);
    o_good: OUT std_logic
  );
END tau;


ARCHITECTURE behavior OF tau IS
  CONSTANT POLY20: std_logic_vector (20 DOWNTO 1) :=
    "11000000011110000000";   --180F & 7*0
  CONSTANT b: integer := 2103;
  CONSTANT c: integer := 1209;
  CONSTANT m: integer := 4096;

BEGIN
  P: PROCESS(clk)
    VARIABLE remainder: std_logic_vector (20 DOWNTO 1);
    VARIABLE crc_asInt: integer range 0 to 4095;
    VARIABLE sign_asInt: integer range 0 to 4095;

  BEGIN
    IF(clk'event AND clk='1') THEN
      remainder := X"FFF00" XOR i_frame(32 DOWNTO 13);

      FOR I IN 20 DOWNTO 1 LOOP
        IF (remainder(20) = '1') THEN
          remainder := remainder XOR POLY20;
        END IF;
        remainder := remainder(19 DOWNTO 1) & '0';
      END LOOP;

      crc_asInt := to_integer(unsigned(remainder(20 DOWNTO 9)));
      sign_asInt := to_integer(unsigned(i_frame(12 DOWNTO 1)));
      IF (((sign_asInt * b) mod m) = ((crc_asInt * c) mod m)) THEN
        o_good <= '1';
      ELSE
        o_good <= '0';
      END IF;

    END IF;
  END PROCESS P;
END behavior;
```

Abbildung 86: Signaturprüfung (VHDL-Code)

Die zur Fehlerinjektion verwendete „Testbench“ ist in Anhang B angegeben. Bevor die zugehörigen Ergebnisse dargestellt werden, sollen – beginnend mit der Realisierung der Schaltung in Hardware – die anderen Modelle dieses Abschnitts beschrieben werden.

Realisierung in Hardware

Um nachzuweisen, dass die oben beschriebene Schaltung nicht nur simuliert, sondern tatsächlich auch in Hardware realisiert werden kann, wurde mit Hilfe des „DE2 Development and Education"-Boards der Firma Altera[19] ein Beispielsystem erstellt, welches die für Entfernte Redundanz benötigten Komponenten in einem FGPA zur Ausführung bringt und über Kippschalter und Anzeigen mit Ein- und Ausgabemöglichkeiten versieht. Das verwendete Board ist in nachfolgender Abbildung 87 dargestellt.



(Altera, 2006)

Abbildung 87: DE2 Development and Education-Board der Firma Altera

Aus der Vielzahl der verfügbaren Bauteile wurde im Rahmen dieser Arbeit nur eine kleine Teilmenge eingesetzt:

- das Altera Cyclone II-FPGA (auf diesem wird die Schaltung ausgeführt)
- die USB-Programmierschnittstelle zum Aufspielen der Schaltung
- das LCD-Display (2x16 Zeichen)
- die Siebensegmentanzeige
- der SRAM-Speicherbaustein (512 KB)
- die Kippschalter
- die Druckknopftaster

[19] http://www.altera.com/

Die Altera-Entwicklungsumgebung liefert die Netzliste eines synthetisierbaren Prozessors („Nios II“) mit, welcher auf dem Cyclone II-FPGA ausgeführt werden kann. Dieser Prozessor wird in der vorliegenden Arbeit zur Kommunikation zwischen den einzelnen Bestandteilen der Schaltung und den Bauteilen des DE2-Boards verwendet und ist die Ausführungsumgebung für ein Anwendungsprogramm, welches es ermöglicht, die Reaktion der Schaltung auf Eingabedaten und Fehlerinjektion zu testen.

Die im Rahmen dieser Arbeit erstellte Anwendung erlaubt es, über die Schalter 0-11 des Boards beliebige Nutzdaten anzulegen. Über diese Daten wird automatisch ein CRC gebildet, welcher ebenfalls automatisch signiert wird (gemäß der Schaltung aus Abbildung 85). Zugleich wird mit jedem neuen Eingabewort ein 8-Bit-Zähler inkrementiert, welcher die Sequenznummer nachbildet. Die senderseitig vorliegenden Daten (Sequenznummer N, Nutzdaten D, CRC C und Signatur S) werden in der oberen Zeile des LCD-Displays als hexadezimale Zahlen angezeigt. Im fehlerfreien Fall gelangen diese Daten unverfälscht zum Empfänger (hier: in eine andere Variable des Programms). Die vom Empfänger verarbeiteten Daten werden nach dem gleichen Schema in der unteren Zeile des LCD-Displays angezeigt, so dass sich insgesamt die folgende Darstellung ergibt:

```
NN DDD (CCC|SSS)
NN DDD (CCC|SSS)
```

Über die Schalter 15-17 können nun drei verschiedene Fehlerfälle eingestellt werden, um die Reaktion der Schaltung auf diese Fehler zu beobachten (vgl. Tabelle 11):

Tabelle 11: Injizierbare Fehler des Anwendungsprogramms

Schalter			**Injizierter Fehler**
17	16	15	
			Fehlerfreier Fall: Die Daten werden unverfälscht zum Empfänger übertragen.
x			Die Nutzdaten werden bei der Übertragung zum Empfänger bitweise invertiert, die Sequenznummer bleibt jedoch unverfälscht.
x	x		Die Nutzdaten werden bei der Übertragung verfälscht und gezielt mit einem dazu passenden CRC versehen.
x	x	x	Anstelle der aktuell anliegenden Daten werden veraltete Daten erneut gesendet.

Empfängerseitig werden nun die tatsächlich erhaltenen Daten geprüft. In diesem Rahmen wird getestet, ob die Daten

- mit der erwarteten Sequenznummer versehen sind,
- zum empfangenen (Klartext-)CRC passen und
- zur empfangenen Signatur passen.

Das Ergebnis der Überprüfung wird in der Siebensegmentanzeige des DE2-Boards angezeigt. Diese Anzeige enthält die Zeichen „CO“ (correct), wenn alle drei Tests bestanden wurden oder „FA“ (false), wenn einer der drei Tests nicht bestanden wurde. Zusätzlich werden die Ergebnisse der drei Tests in oben genannter Reihenfolge angezeigt, also etwa:

```
FA 00 01 01
```

wenn ausschließlich die Sequenznummer als ungültig erkannt wurde oder aber im fehlerfreien Fall:

```
CO 01 01 01
```

sofern alle drei Tests bestanden wurden. Es sei darauf hingewiesen, dass die hier realisierte Fehlerinjektion im Gegensatz zu anderen im Rahmen dieser Arbeit betrachteten Testverfahren in erster Linie zu Demonstrations- und Lehrzwecken dient und dementsprechend einfach gehalten ist. Ferner würde eine Übertragung des CRC im Klartext – zusätzlich zur Signatur – normalerweise nicht erfolgen und ist hier ebenso zu Anschauungszwecken realisiert. Das in diesem Abschnitt beschriebene Programm ist im elektronischen Anhang der Arbeit enthalten sowie in Anhang C aufgeführt.

Vollständiger Demonstrator

Basierend auf den Ergebnissen des vorangegangenen Kapitels entstand in einem vom Autor betreuten Studienprojekt (Petroff, 2009) ein erweiterter Versuchsaufbau, bestehend aus nunmehr zwei DE2-Boards und zwei FlexRay-Knoten, um die komplette Strecke von der Erzeugung und Signatur eines Sensorwertes über die Weiterleitung durch das Bussystem bis zur abschließenden Signaturprüfung und Verwendung der Daten beim empfangenden Aktuator nachzubilden.

Das Grundgerüst dieses Aufbaus war Basis für den im Folgenden beschriebenen Demonstrator (s. Abbildung 88) und die mit ihm durchgeführten Untersuchungen. Die realisierte Funktion lässt sich dabei wie folgt beschreiben:

Sensorwerte werden als vom Programm zufällig erzeugte Eingaben von einem der beiden DE2-Boards generiert. Diese Daten werden (ggf. unter Fehlerinjektion) signiert, mit einer Sequenznummer versehen und per SPI[20] an das darunter befindliche FlexRay-Board gesendet. Von dort werden sie über das Bussystem an das andere FlexRay-Board übertragen und gelangen dann wiederum per SPI an das zweite DE2-Board, welches den Empfänger darstellt. Hier werden die empfangenen Daten hinsichtlich ihrer Signatur und Sequenznummer geprüft und schließlich zusammen mit dem Ergebnis dieser Prüfung angezeigt.

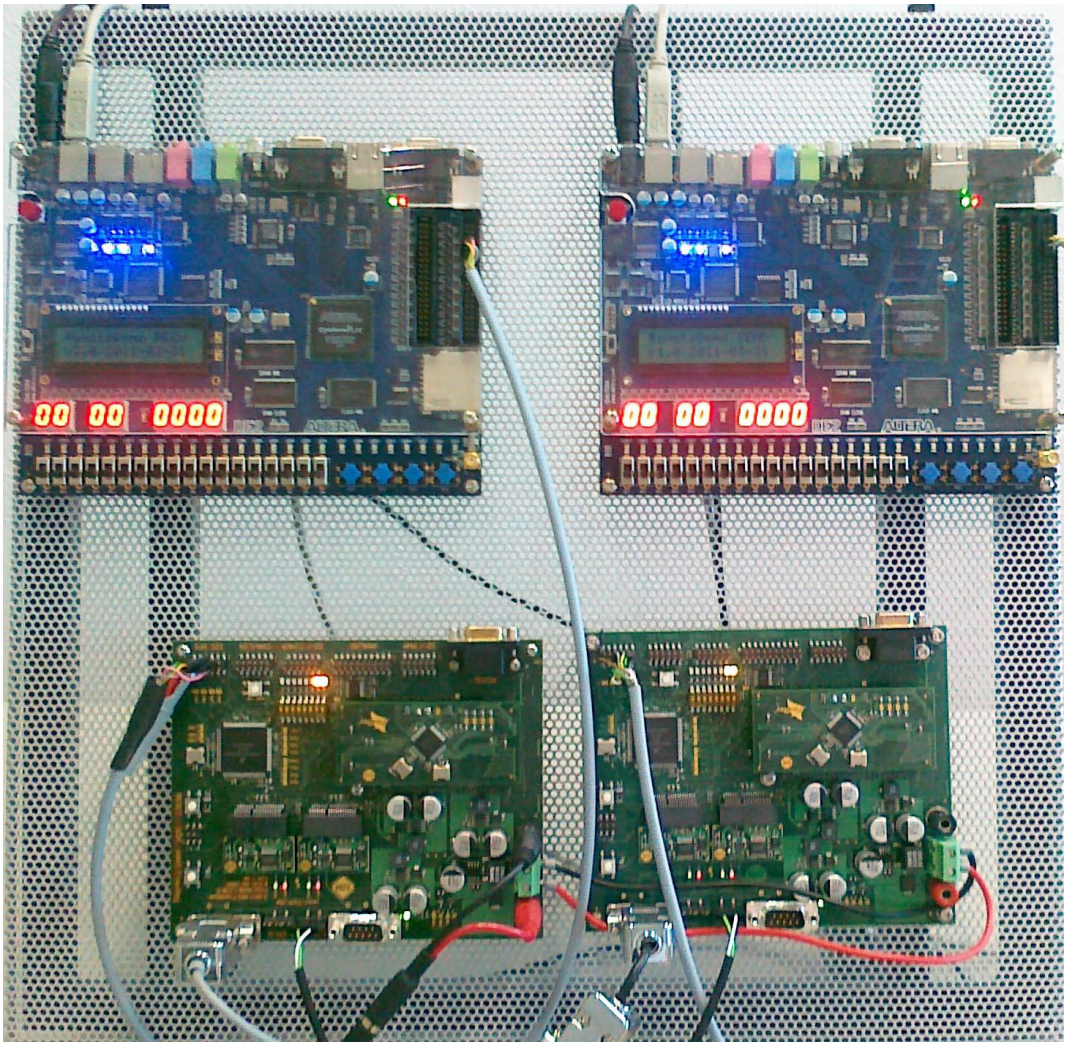


Abbildung 88: Vollständiger Demonstrator

[20] Serial Peripheral Interface

Im Mittelpunkt der Untersuchung stand nun allerdings nicht mehr die bereits betrachtete Signaturprüfung, sondern die Verwendung von Sequenznummern mit dem in Kapitel 4.4 skizzierten Verfahren der Sequenznummernzerlegung aus (Echtle & Kimmeskamp, 2009). Dieses Verfahren wurde für die vorliegende Arbeit in ebenfalls VHDL implementiert und ist in nachfolgender Abbildung 89 am Beispiel einer insgesamt 36 Bit langen Sequenznummer dargestellt.

```
loop
  -- bei Fehler auf  zuletzt  empfangenen Daten aufsetzen
  if not error then
    empf := lese_bus;
    n0 := empf(3 downto 0);
    ni := empf(7 downto 4);
    i := n0 mod 8 + 1;
  end if

  -- lokale SeqNr aktualisieren
  if sync = 0 then -- lokale SeqNr komplett neu aufbauen
    seqnr(3 downto 0) := n0;
    seqnr((i+1)*4-1 downto (i*4)) := ni;
    sync++;
  else if sync = 8 -- lokale SeqNr im Ganzen inkrementieren
    seqnr++ (mod 2^36);
  else -- niedr. 4 Bit der lokalen SeqNr inkrementieren, ni einfügen
    seqnr(3 downto 0)++ (mod 16);
    seqnr((i+1)*4-1 downto (i*4)) := ni;

    carry := (seqnr(3 downto 0) = 0 or carry) and seqnr((i+1)*4-1 downto (i*4)) = 0;
    if sync = 7 then -- ggf. Übertrag durchführen
      if carry then
        seqnr(35 downto (i+1)*4)++;
        carry := false;
      end if
    end if

    sync++;
  end if

  -- empfangene Daten mit lokaler SeqNr vergleichen
  if sync < 8 then
    error := seqnr(3 downto 0) != n0;
  else
    error := (seqnr(3 downto 0) != n0) or (seqnr((i+1)*4-1 downto (i*4) != ni);
  end if

  -- auf ungültige SeqNr reagieren
  if error then
    errno++;   -- Fehler zählen
    sync := 0; -- bei jedem Fehler neu synchronisieren
  end if
end loop
```

Abbildung 89: Aufbau von Sequenznummern (vereinfachtes VHDL)

Die gesamte Sequenznummer wird zur Übertragung in neun Blöcke $n_8, n_7, \ldots, n_0$ zu je 4 Bit aufgeteilt. Die im Block n_0 enthaltenen niederwertigsten Stellen werden dabei stets übertragen. Zusätzlich übertragen wird jeweils Block n_i, mit $i = n_0 \, mod \, 8 + 1$. Der Empfänger erhält also in jedem Zyklus insgesamt 8 Bit, aus denen die lokale Sequenznummer nun sukzessive aufgebaut werden muss.

Zunächst werden, sofern kein Fehler vorliegt, die beiden Anteile n_i und n_0 über den Bus empfangen. Anschließend sind drei Fälle zu unterscheiden: Wenn beim Empfänger lokal noch keine Informationen zur Sequenznummer vorliegen (`sync = 0`), so muss der vollständige Neuaufbau der Sequenznummer initiiert werden. Hierzu sind zunächst die beiden empfangenen Blöcke an die zugehörigen Stellen der Sequenznummer einzutragen. Wenn bereits alle Blöcke vorliegen (`sync = 7`), kann hingegen die lokale Sequenznummer im Ganzen inkrementiert werden. In der dazwischenliegenden Phase schließlich werden die niederwertigsten 4 Bits der Sequenznummer bereits inkrementiert, während die noch zu empfangenden Blöcke n_i in die im Aufbau befindliche Nummer eingetragen werden. Dabei ist ein Übertrag zu berücksichtigen, falls die Grenze des verwendeten Zahlenbereichs erreicht wird, da bereits eingefügte Teilstücke nun ggf. nicht mehr aktuell sein können.

Die Prüfung der empfangenen Daten unterscheidet sich ebenfalls nach der gerade durchgeführten Phase: Unmittelbar beim Neuaufbau der Sequenznummer kann noch keine Prüfung vorgenommen werden, da lokal noch keine Daten für einen Vergleich vorliegen. Während des sukzessiven Aufbaus der Nummer können bereits die niederwertigsten 4 Bits der lokalen Sequenznummer mit dem empfangenen Block n_0 verglichen werden. Sobald die Sequenznummer vollständig vorliegt, können beide empfangenen Anteile n_i und n_0 mit den entsprechenden Anteilen der lokalen Sequenznummer verglichen werden.

Ein erkannter Fehler führt in der für diese Arbeit implementierten Anwendung zur Inkrementierung eines Fehlerzählers und zum Wiederaufsetzen auf der neu erhaltenen Sequenznummer.

Für den gesamten Demonstrator ergibt sich insgesamt der folgende Ablauf: Zufällige Sensordaten werden im fehlerfreien Fall mit einer fortlaufend inkrementieren Sequenznummer versendet. Per Knopfdruck kann ein Bündelfehler eingestreut werden, wobei die Sequenznummer jedoch explizit ausgenommen ist. Ebenfalls auf Knopfdruck kann die Sequenznummer des Senders zufällig neu gesetzt werden, was beim Empfänger das oben beschriebene Verfahren zum Aufbau der Sequenznummer auslöst. Die vollständigen senderseitigen Eingabemöglichkeiten des Demonstrators sind in Tabelle 12 aufgeführt.

Tabelle 12: Eingabemöglichkeiten des Demonstrators

Taster	Aktion
Key3	Start/Pause der Datenübertragung
Key2	Bündelfehler einstreuen (ohne Sequenznummer)
Key1	Zufällige neue Sequenznummer setzen
Key0	Reset der Anwendung

Beim Empfänger wird die Anzahl der erkannten Fehler (getrennt nach Fehlerart) in der Siebensegmentanzeige protokolliert, vollständige Log-Daten werden optional an einen über USB angeschlossenen Laptop übermittelt. Mittels der LEDs des Empfänger-Boards kann zudem der schrittweise Aufbau der Sequenznummer nachverfolgt werden.

Die vollständige VHDL-Implementierung des hier beschriebenen Verfahrens befindet sich im elektronischen Anhang der Arbeit, die Kommunikation der Flexray-Knoten ist in Anhang D enthalten.

5.2.4.3 Ergebnis

VHDL-Implementierung

Das im Rahmen dieser Arbeit verwendete Signaturschema basiert auf einer herkömmlichen CRC-Berechnung und erweitert diese um die Multiplikation mit einem geheimen Faktor, um nicht nur zufällige Fehler aufdecken, sondern Integrität und Authentizität empfangener Nachrichten auch bei mutwilligen Manipulationen überprüfen zu können.

Es lassen sich mit Hilfe von CRCs alle Bündelfehler bis zur Länge $Grad\ g(x)$ erkennen, wobei $g(x)$ das dem verwendeten CRC zugrundeliegende Generatorpolynom ist. Ein Bündelfehler $e(x)$ der Länge t ist dabei definiert als Polynom $x^i b(x)$, mit $Grad\ b(x) < t$ und $0 \leq i \leq n - t$, d. h. $e(x)$ ist an höchstens t aufeinanderfolgenden Stellen ungleich Null (Friedrichs, 1995).

Um nun zu untersuchen, inwiefern diese Eigenschaft auch bei zusätzlicher Anwendung des Signaturverfahrens erhalten bleibt, wurde, wie bereits beschrieben, die realisierte Schaltung entsprechenden Testläufen unterworfen. Hierzu wurden Simulationsläufe für jeweils 10.000 zufällig generierte Nutzdatenwörter unter Injizierung o. g. Fehlerart mit dem Wert `0x28F` und allen in Tabelle 9 dargestellten Parametersätzen durchgeführt.

Das verwendete Werkzeug ModelSim erlaubt es, die Ergebnisse der Schaltungssimulation im Zeitverlauf als sog. „Waveform“ darzustellen. Am Beispiel eines konkreten Zeitpunktes (s. Abbildung 90) soll der Versuchsaufbau beschrieben werden:

Das Taktsignal `clk` besitzt eine Periode von 10 ms, alle Daten werden stets bei steigender Taktflanke übernommen. Zunächst sind dies die im Signal `nextdata` enthaltenen, zufällig generierten Nutzdaten. Im nächsten Takt liegen diese, erweitert um die vom Sender hierüber gebildete Signatur, als vollständige Nachricht `frame_s` vor. Die Nachricht wird unter Fehlerinjektion übertragen und gelangt so als `frame_r` zum Empfänger. Dort schließlich wird die Signatur der Nachricht geprüft und das Ergebnis der Prüfung im Diagnosesignal `good` bereitgestellt. Mitgeführt werden in der Simulation die im aktuellen Takt sowie in den beiden vorhergehenden Takten injizierten Fehler (`error`, `erro1` und `erro2`), da insgesamt, wie oben geschildert, drei Takte notwendig sind, um die ursprünglichen Nutzdaten mit der Signatur zu versehen, die Nachricht zu „senden“ und schließlich die Signatur zu prüfen. Es ist folglich stets `erro2` der Fehler, welcher in die zuletzt überprüfte Nachricht injiziert worden war.

Für den in der Abbildung dargestellten Takt ist daher das Signal `good` logisch 1, da der durch `erro2` dargestellte Bündelfehler zufällig kein „echter“ Fehler ist. Würde der injizierte Fehler nicht aus lauter Nullen bestehen und wäre das Diagnosesignal trotzdem logisch 1, so würde ein unerkannter Fehler vorliegen.

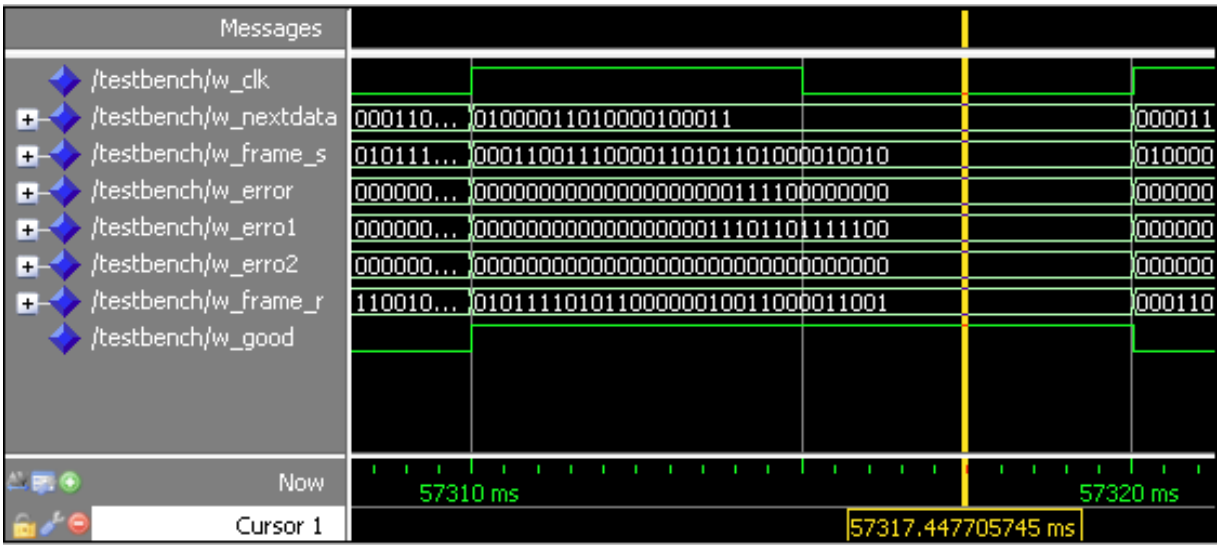


Abbildung 90: Schaltungssimulation (Waveform)

Zusätzlich zur Analyse der erzeugten Waveforms anhand von Zusicherungen erlaubt es ModelSim, die Ergebnisse in eine Datei zu schreiben, so dass auch bei einem großen Stichprobenumfang die Anzahl unerkannter Fehler nachvollzogen werden kann. Die Ergebnisse der Überprüfung der im Rahmen dieser Arbeit durchgeführten Simulationsläufe sind in nachfolgender Abbildung 91 dargestellt.

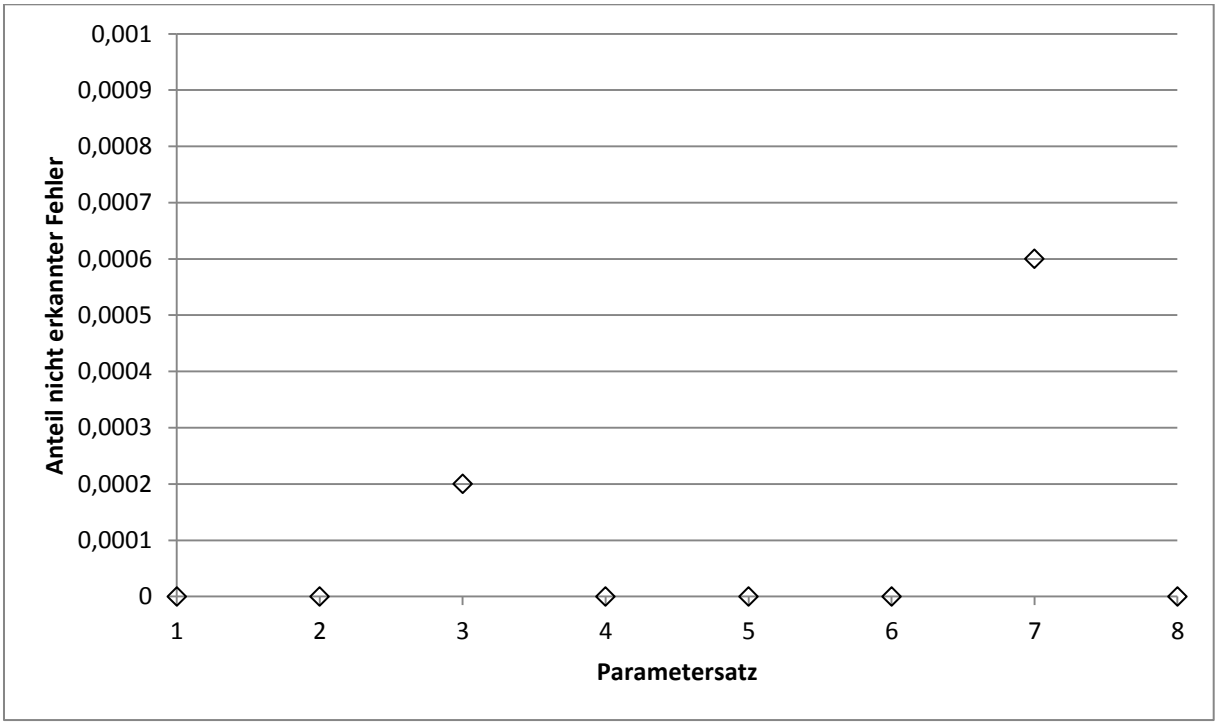


Abbildung 91: Ergebnis der Schaltungssimulation

Mit Ausnahme des dritten und sechsten Parametersatzes wurden alle Fehler injizierten Bündelfehler wie erwartet erkannt. Die beiden genannten Parametersätze schnitten schlechter ab als der zugrundeliegende CRC und wiesen einen Anteil nicht erkannter Fehler von 0,0002 bzw. 0,0006 auf.

Bei genauerer Betrachtung der zugehörigen Parameter fällt auf, dass ausschließlich in beiden genannten Fällen der zufällig gewählte geheime Schlüssel `a` eine gerade Zahl ist. Die Vermutung liegt also nahe, dass ein solcher Schlüssel für das Signaturverfahren ungünstig ist. Tatsächlich gibt es hierfür eine mathematische Erklärung: Da gerade Zahlen in Dualdarstellung stets mit einer Null enden und hierdurch auch die niederwertigste Stelle des Produkts stets eine Null ist, trägt diese nicht zur Information bei. De facto verkürzt sich also die Schlüssellänge gegenüber dem CRC um eine Stelle.

Als Ergebnis lässt sich somit festhalten, dass das betrachtete Signaturverfahren eine zu CRCs vergleichbare Güte aufweisen kann, wenn die verwendeten Schlüssel gut gewählt sind. Wird dies berücksichtigt, so bietet es bei einem nur geringen Zusatzaufwand durch die Gewährleistung von Authentizität einen zusätzlichen Nutzen gegenüber dem der Berechnung zugrundeliegenden CRC.

Es sei an dieser Stelle erneut darauf hingewiesen, dass Entfernte Redundanz generisch bezüglich des verwendeten Signaturverfahrens ist. Der vorangegangene Abschnitt stellt also keine Bewertung Entfernter Redundanz dar, sondern lediglich eine Bewertung des hier verwendeten Verfahrens.

Realisierung in Hardware

Wie bereits beschrieben, diente die Hardwarerealisierung der Signaturerzeugung und -prüfung in erster Linie als Machbarkeitsstudie. Die Berechnung von CRCs lässt sich bekanntermaßen mit Hilfe rückgekoppelter Schieberegister realisieren, was exakt dem weiter oben angegebenen Algorithmus entspricht (Lang, 2011). Alternativ ist es möglich, bereits bei Systemerzeugung eine Tabelle zu erstellen, welche für jedes zu übertragende Byte den zugehörigen CRC-Wert enthält, so dass dieser zur Laufzeit lediglich in der Tabelle nachgeschlagen werden muss. Das in dieser Arbeit verwendete Synthesetool der Quartus II-Entwicklungsumgebung hat, wie im Ausschnitt aus Abbildung 92 ersichtlich, die CRC-Berechnung aus mehreren hintereinandergeschalteten Multiplexern realisiert, was ebenfalls eine unmittelbare Bereitstellung des Ergebnisses ermöglicht (zwei Takte werden für die Datenübernahme benötigt).

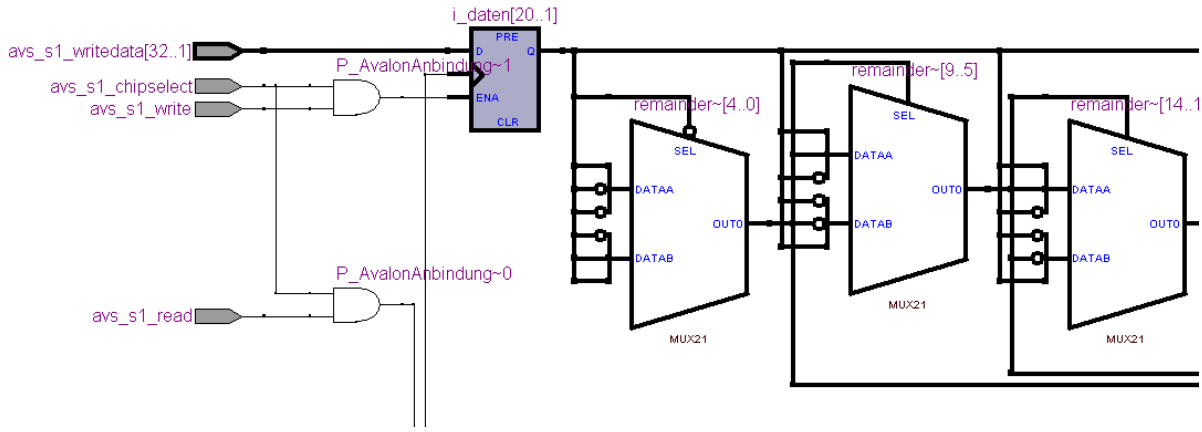


Abbildung 92: CRC-Berechnung (Ausschnitt)

Die Signaturerzeugung kann auf dem FPGA, wie in Abbildung 93 dargestellt, mit einem der dort verfügbaren Multiplizierern realisiert werden. Auf diese Weise lässt sich die Berechnung innerhalb eines Systemtaktes ausführen, wobei erneut zwei Takte für die Datenübernahme benötigt werden.

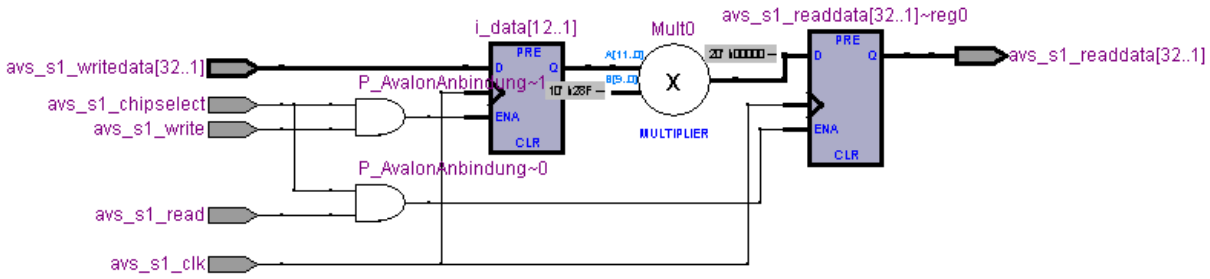


Abbildung 93: Signaturerzeugung

Ähnlich kann die Signaturprüfung umgesetzt werden (s. Abbildung 94). Die beiden Multiplikationen können parallel ausgeführt werden, zusätzlich wird lediglich ein Komparator benötigt. Die Datenübernahme erfordert wiederum zwei Systemtakte.

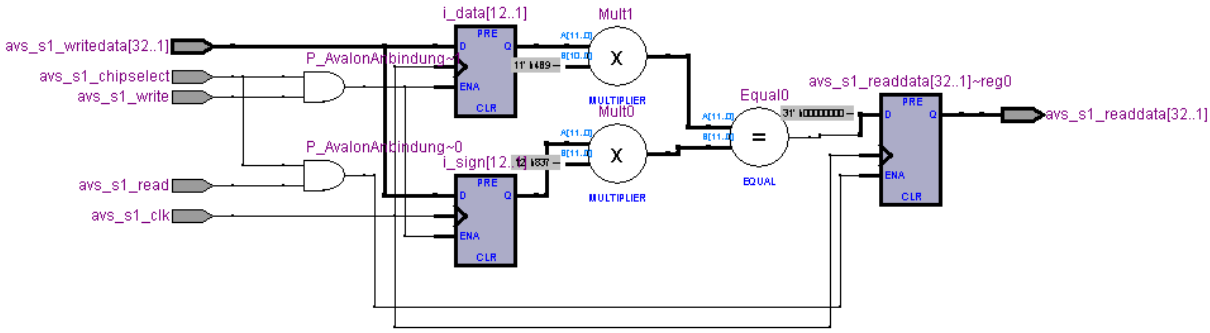


Abbildung 94: Signaturprüfung

Insgesamt konnte hiermit nachgewiesen werden, dass sich die für Entfernte Redundanz erforderlichen Komponenten mit geringem Aufwand in Hardware realisieren lassen und keinen relevanten Overhead bzgl. der für die Berechnungen benötigten Zeit darstellen.

Es sei dabei erneut darauf hingewiesen, dass hier lediglich ein mögliches Signaturverfahren dargestellt wurde. Abhängig von den Anforderungen der Anwendung können im Rahmen von Entfernter Redundanz ebenso beliebig aufwändigere oder einfachere Verfahren verwendet werden.

Tabelle 13: Reaktion der Schaltung auf injizierte Fehler

Injizierter Fehler	Reaktion beim Empfänger
Kein Fehler	CRC, Signatur und Sequenznummer sind gültig und werden als gültig erkannt.
Bitweise Invertierung der Nutzdaten	CRC und damit auch Signatur passen nicht mehr zu den Nutzdaten und werden vom Empfänger als ungültig erkannt.
Mutwillige Fälschung der Nutzdaten und Berechnung eines dazu passenden CRCs	Der Empfänger kann diesen Fehler nicht mehr am CRC, jedoch weiterhin anhand der ungültigen Signatur (welche nach Annahme nicht gefälscht werden kann) aufdecken.
Wiederholtes Senden veralteter Daten	Sowohl CRC als auch Signatur passen in diesem Fall zur Nachricht, der Empfänger erkennt jedoch anhand der veralteten Sequenznummer den Fehler.

Obenstehende Tabelle 13 zeigt abschließend die Reaktion der in Hardware realisierten Schaltung auf Fehler. Wie bereits beschrieben, wurden hier zu Demonstrationszwecken verschiedene Szenarien implementiert. Anhand dieser Beispiele lässt sich folgender Zusammenhang nachvollziehen: während die zufällige Verfälschung von Nutzdaten bereits durch den CRC erkannt werden, lassen sich gezielte Angriffe, welche auch die Nachbildung eines CRC einschließen können, nur durch kryptografische Verfahren verhindern. Werden durch einen Fehler oder im Rahmen eines Angriffs wiederholt alte Daten gesendet (zu denen ja bereits eine gültige Signatur vorliegt), so lässt sich dies durch die Verwendung von Sequenznummern aufdecken.

Vollständiger Demonstrator

Auch der Aufbau eines Hardwaredemonstrators diente zum einen als Machbarkeitsstudie. In diesem Sinne konnte nachgewiesen werden, dass in einem realistischen Szenario mit Datenübertragung über ein physikalisch vorhandenes Bussystem die für Entfernte Redundanz erforderliche Funktionalität in zweckmäßiger Weise erbracht werden kann. Zum anderen konnte durch die oben bereits erwähnte Möglichkeit der Fehlerinjektion erneut erfolgreich die Eignung des in Kapitel 4.2 beschriebenen Signaturverfahrens demonstriert werden. Die bereits im Rahmen der Schaltungssimulation verwendete Klasse der Bündelfehler wurde hierfür erstmalig in die reale Schaltung injiziert.

Besonderes Untersuchungsobjekt im vorliegenden Arbeitsabschnitt war jedoch die Methode der Sequenznummernzerlegung aus Kapitel 4.4. Sie wurde für den Demonstrator erfolgreich als Schaltung implementiert und ebenso einer Fehlerinjektion unterworfen. Diese bestand hier darin, dass die Sequenznummer im Sender nicht wie im fehlerfreien Fall inkrementiert, sondern auf einen zufälligen neuen Wert gesetzt wurde, mit welchem das Verfahren dann bis zur Injektion des nächsten Fehlers planmäßig fortgesetzt wurde.

Durch die empfängerseitige Ausgabe entsprechender Log-Dateien konnte der folgende Sachverhalt beobachtet werden:

- In der Initialisierungsphase werden auf die niederwertigsten 4 Bits beschränkte Fehler stets erkannt. Die Erkennung weiterer Fehler kann nicht garantiert werden.
- Sobald die Sequenznummer beim Empfänger komplett vorliegt, werden beliebige Fehler erkannt, es können jedoch maximal acht Zyklen bis zur Erkennung verstreichen.

Diese Eigenschaften entsprechen genau den an den Algorithmus gestellten Erwartungen, wobei es naturgemäß von der (hier nicht betrachteten) Anwendung abhängen wird, ob dies als geeignet einzustufen ist. Aus selbigem Grund war die Reaktion der Anwendung auf erkannte Fehler ebenfalls nicht Gegenstand der Untersuchung (die Fehler wurden lediglich gezählt und der Aufbau der Sequenznummer neu initialisiert).

Die in Abbildung 95 dargestellten Beispiele erläutern einige typische beobachtbare Szenarien. Die Zeilen 1-35 stellen den fortschreitenden Zeitverlauf dar; die Spalten A-I stehen für die empfängerseitig bereits vorliegenden Anteile $n_8, n_7 \ldots, n_0$ der Sequenznummer. Senderseitig liegt anfangs (Zeile 1) als Sequenznummer der Wert 0 an, d. h. die jeweils 4 Bit großen Anteile der vollständigen Sequenznummer lassen sich in hexadezimaler Schreibweise als `0x000000000` darstellen. Der Empfänger erhält nun zunächst das Wort `0x00` und verwendet es, um n_0 und n_1 *seines* Zählers entsprechend zu belegen. Anschließend werden `0x01`, `0x02`, `0x03`, usw. empfangen, wobei die für Anteil n_0 stehenden niederwertigsten 4 Bits bereits mit dem entsprechenden Teil der bis dahin lokal vorliegenden Sequenznummer verglichen werden können. Die für die Anteile $n_2, n_3, \ldots, n_8$ stehenden höherwertigsten 4 Bits werden vom Empfänger sukzessive zum Aufbau der Sequenznummer verwendet. In Zeile 8 ist die Sequenznummer vollständig (sie hat nun den Wert `0x000000007`). Fortan werden beide Teile des vom Empfänger erhaltenen Datenwortes mit der lokal vorliegenden Sequenznummer verglichen.

	A	B	C	D	E	F	G	H	I
1	?	?	?	?	?	?	?	0	0
2	?	?	?	?	?	?	0	0	1
3	?	?	?	?	?	0	0	0	2
4	?	?	?	?	0	0	0	0	3
5	?	?	?	0	0	0	0	0	4
6	?	?	0	0	0	0	0	0	5
7	?	0	0	0	0	0	0	0	6
8	0	0	0	0	0	0	0	0	7
9	0	0	0	0	0	0	0	0	8
10	0	0	0	0	0	0	0	0	9
11	0	0	0	0	0	0	0	0	A
12	0	0	0	0	0	0	0	0	B
13	?	?	?	?	?	0	?	?	2
14	?	?	?	?	0	0	?	?	3
15	?	?	?	0	0	0	?	?	4
16	?	?	0	0	0	0	?	?	5
17	?	0	0	0	0	0	?	?	6
18	0	0	0	0	0	0	?	?	7
19	0	0	0	0	0	0	?	0	8
20	0	0	0	0	0	0	0	0	9
21	0	0	0	0	0	0	0	0	A
22	0	0	0	0	0	0	0	0	B
23	0	0	0	0	0	0	0	0	C
24	0	0	0	0	0	0	0	0	D
25	0	0	0	0	0	0	0	0	E
26	0	0	0	0	0	0	0	0	F
27	?	?	?	?	?	?	?	2	0
28	?	?	?	?	F	?	?	?	B
29	?	?	?	F	F	?	?	?	C
30	?	?	F	F	F	?	?	?	D
31	?	F	F	F	F	?	?	?	E
32	E	F	F	F	F	?	?	?	F
33	E	F	F	F	F	?	?	0	0
34	E	F	F	F	F	?	0	0	1
35	F	0	0	0	0	0	0	0	2

Abbildung 95: Aufbau der Sequenznummer beim Empfänger

In Zeile 13 tritt nun senderseitig ein Fehler auf: Die Sequenznummer springt auf `0x000000002` zurück. Dieser Fehler wird unmittelbar erkannt, da der Empfänger für n_0 an dieser Stelle den Wert `0xC` (anstelle von `0x2`) erwartet. Er beginnt folglich damit, die Sequenznummer auf Basis der gesamten in diesem Zyklus erhaltenen Daten, also `0x02`, neu aufzubauen. Da der für n_0 erhaltene Wert bestimmt, welcher Anteil der Sequenznummer aus den höherwertigsten 4 Bits der erhaltenen Daten belegt wird, beginnt hier der Aufbau der Sequenznummer bei n_3.

In Zeile 20 ist die Sequenznummer beim Empfänger wieder vollständig, sie hat jetzt den Wert `0x000000009`. Zur selben Zeit tritt jedoch erneut ein Fehler auf, die senderseitige

Sequenznummer springt auf den Wert `0x000000019`. Dieser Fehler wird beim Empfänger zunächst nicht erkannt, da der n_1 entsprechende Anteil erst in Zeile 27 wieder mit der lokal vorliegenden Sequenznummer verglichen wird. Da nun vom Sender aufgrund der dort geänderten Sequenznummer der Wert `0x20` empfangen wird (und nicht wie erwartet `0x10`), zeigt sich nun, dass ein Fehler aufgetreten sein muss und die Sequenznummer ist vom Empfänger vollständig neu aufzubauen.

Bevor dies geschehen kann, springt die Sequenznummer des Senders jedoch bereits im nächsten Zyklus auf den Wert `0xEFFFFFFFB`. Dies wird vom Empfänger unmittelbar erkannt, da ein Sprung in n_0 vorliegt (erwartet wird `0x1`, empfangen wird `0xB`). Der Aufbau der Sequenznummer ist also erneut anzustoßen. Eine Besonderheit zeigt sich dabei ab Zeile 32: Da für einen Teil der Sequenznummer der verfügbare Zahlenbereich überschritten wird, tritt senderseitig beim Wechsel auf Zeile 33 ein Übertrag auf (von `0xEFFFFFFFF` auf `0xF00000000`). In Zeile 35 ist dieser Übertrag vom Empfänger für die noch nicht im Zuge des Aufbaus der Sequenznummer aktualisierten Anteile $n_8, n_7 \dots, n_4$ wie abgebildet nachzuholen.

Es sei abschließend darauf hingewiesen, dass es in keiner Weise erforderlich ist, die hier beschriebene Technik der Sequenznummernzerlegung bei der Verwendung Entfernter Redundanz anzuwenden. Sie stellt lediglich eine Möglichkeit dar, deren Nutzen vom konkreten Anwendungsfall abhängen wird. Gezeigt werden konnte jedoch, dass es denkbar und möglich ist, auf diese Weise den Wertebereich der Sequenznummer erheblich zu vergrößern, ohne hierdurch zusätzliche Bandbreite bei der Datenübertragung erforderlich zu machen.

5.3 Vorläufiges Resümee

Bevor im nachfolgenden Kapitel 6 das mögliche Potential Entfernter Redundanz aus verschiedenen Blickwinkeln betrachtet wird, sollen zunächst die aus der Untersuchung des Beispielsystems in diesem Teil der Arbeit gewonnenen Ergebnisse zusammengefasst werden.

- Eine vergleichende Fehlerbaumanalyse hat ergeben, dass die Systemvariante der elektronisch geregelten Lenkung mit Entfernter Redundanz weniger minimale Schnitte aufweist als ein entsprechendes System mit Dedizierter Redundanz. Mehr noch: Die Schnitte bilden eine echte Teilmenge der Schnitte, die bei Dedizierter Redundanz entstehen. Das System mit Entfernter Redundanz weist also stets eine höhere Zuverlässigkeit auf, wenn die verwendeten Komponenten selbst nicht

unzuverlässiger sind als bei Dedizierter Redundanz. Bei Berechnungen anhand realistischer Fehlerraten ergaben sich praktisch identische Ergebnisse.

- Die formale Verifikation eines noch relativ stark vereinfachten Systemmodells mit UPPAAL hat gezeigt, dass alle in der Fehlervorgabe enthaltenen Einzelfehler wie gefordert toleriert werden. Für alle Haftfehler konnte darüber hinaus formal nachgewiesen werden, dass die vorgesehenen Maßnahmen zur Fehlerbehandlung im Fehlerfall erfolgreich durchgeführt werden und im fehlerfreien Fall ausbleiben.
- Fehlerinjektionsexperimente im Rahmen einer detaillierteren funktionalen Simulation mit MATLAB/Simulink haben diese Ergebnisse bestätigt. Zugleich konnte demonstriert werden, wie die Funktion der für Entfernte Redundanz erforderlichen Komponenten programmtechnisch realisiert werden kann und wie sich die durch Entfernte Redundanz verursachte zusätzliche Kommunikation der Komponenten untereinander über ein (simuliertes) Bussystem gestaltet.
- Die prototypische Implementierung in VHDL konnte zeigen, dass das in dieser Arbeit verwendete Signaturschema bei geeigneter Wahl der Parameter sachgerecht arbeitet, nicht schlechter abschneidet als das zugrundeliegende CRC-Verfahren und in für den industriellen Einsatz der betrachteten Anwendung typischer Hardware realisierbar ist. Ein vollständiger Demonstrator implementierte neben dem Signaturverfahren und der Kommunikation über ein reales Bussystem erfolgreich ein Verfahren zur Steigerung der Effizienz bei der Verwendung von Sequenznummern.

Auf allen betrachteten Abstraktionsebenen ließ sich somit das Konzept Entfernter Redundanz am Beispielsystem der elektronisch geregelten Lenkung umsetzen und zeigte in jedem Fall die vorab vermutete Eigenschaft: Beibehaltung der geforderten Fehlertoleranz bei geringerem Aufwand.

6 Bewertung des Potenzials der Idee

6.1 Bedeutung für die Fehlertoleranzforschung

Als Erweiterung der in Kapitel 3 dieser Arbeit präsentierten Übersicht zum Redundanzbegriff (Abbildung 2) kann Entfernte Redundanz zunächst als Ausgangspunkt einer neuen Beschreibungsdimension angesehen werden, zusätzlich zum verwendeten redundanten Mittel (strukturelle, funktionelle, Informations- und Zeitredundanz) und zur Aktivierung der Redundanz (statisch, dynamisch, hybrid): Ist ein redundantes Mittel ausschließlich für unmittelbar mit ihm verbundene Komponenten verfügbar, stellt es Dedizierte Redundanz dar, andernfalls wird es als Entfernte Redundanz bezeichnet. Die sich hieraus ergebende Taxonomie ist in nachfolgender Abbildung 96 dargestellt.

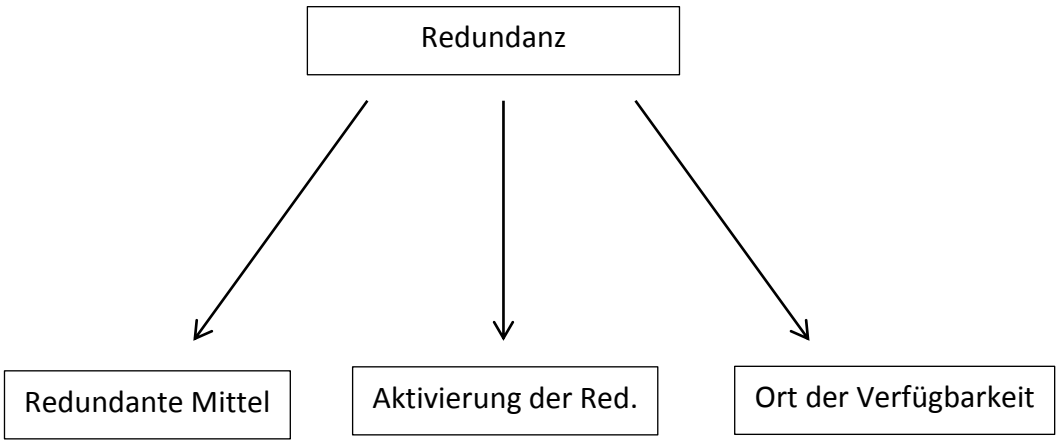


Abbildung 96: Erweiterter Redundanzbegriff

Eng verbunden mit der Schaffung dieser neuen Beschreibungsdimension ist der mögliche Nutzen für die praktische Fehlertoleranzforschung. Wie bereits in Kapitel 1 der Arbeit ausgeführt, hatte diese es schon in der Vergangenheit zunehmend mehr zum Ziel, den Aufwand für die Erbringung von Fehlertoleranz zu minimieren. Typischerweise wurde versucht, durch geschickte Kombination oder Substitution verschiedener Redundanzformen einen gegebenen Grad an Fehlertoleranz beizubehalten, die hierfür entstehenden Kosten jedoch zu verringern. Das Konzept der Entfernten Redundanz schafft dabei durch den neu geschaffenen Freiheitsgrad zusätzliche Möglichkeiten. Gedankliche Wurzeln lassen sich jedoch durchaus in bereits existierenden Arbeiten finden. Zu nennen sind hier insbesondere die Konzepte „Time-Staggered Redundancy“ und „Distance Agreement Protocols“, virtuelle Duplex-

systeme, Doppelduplexsysteme und gegenseitige Redundanz. Im Vergleich mit diesen Ansätzen soll nun der zusätzliche Beitrag Entfernter Redundanz aufgezeigt werden, um das Konzept zugleich in die bereits existierende Forschung einzuordnen.

Beim Konzept „Time-Staggered Redundancy“ (Echtle, 1987) wird ein Prozess zunächst nur in n Knoten redundant ausgeführt. In $m-n$ weiteren Knoten wird er mit einer einstellbaren Zeitverzögerung redundant ausgeführt. Damit eröffnet sich die Möglichkeit, temporär falsche Eingaben (z. B. durch kurzzeitige „Blendung“ aller Sensoren) zu tolerieren. Nachdem Absoluttests die Fehleingaben erkannt haben, kann mit den $m-n$ zeitverzögerten Exemplaren des Prozesses weitergearbeitet werden. Wenn alle Prozesse eines Prozesssystems zeitgestaffelt ausgeführt werden und die Verarbeitung als Reaktion auf Eingaben in Echtzeit erfolgt, kann später wieder auf die n primären, nicht zeitverzögerten Prozesse zurückgegangen werden.

In „Distance Agreement Protocols“ (Echtle, 1989) verläuft ein „Pendelschlag“ wie folgt: n Knoten bilden Signaturen für einen vorgeschlagenen Wert, um eine fehlerfreie Mehrheit festzustellen. Dabei werden die n Knoten sequentiell durchlaufen. Wird die Mehrheit festgestellt, dann wird der n-fach signierte Wert an alle verbleibenden $m-n$ Knoten per Multicast gesandt.

Auch bei Entfernter Redundanz findet sich die Aufteilung eines n-von-m-Systems im Verhältnis $n:(m-n)$, wobei n Knoten die „Hauptarbeit“ verrichten, beispielsweise indem sie Motoren ansteuern, und $m-n$ Knoten der Überwachung dienen. Im Gegensatz zu den oben genannten Ansätzen wurde bei Entfernter Redundanz jedoch speziell die Verdrahtungstopologie zwischen Rechnern und Peripheriegeräten betrachtet. Dabei zeigte sich, dass eine Überwachung (bei Verwendung entsprechender Schutzmaßnahmen) auch über fremde Knoten hinweg möglich ist. Im Zuge der naheliegenden Idee einer gemeinsamen Nutzung von Rechenkapazitäten konnte somit ein für die Fehlertoleranz charakteristisches Prinzip derart aufgegriffen werden, dass schließlich nur noch n Knoten physikalisch existieren müssen, während die Funktion der übrigen $m-n$ Knoten vollständig auf Software abgebildet werden kann.

Virtuelle Duplexsysteme (Jochim, 2003) ersetzen strukturelle Redundanz durch Zeitredundanz. Anstelle einer gleichzeitigen Ausführung der betrachteten Funktion auf mehreren Komponenten, tritt die mehrmalige Ausführung auf derselben Komponente, in der Erwartung, dass sich ein Fehler nur zeitlich begrenzt auswirkt und somit bei zweimaliger

Ausführung durch Vergleich beider Ergebnisse erkannt oder bei mindestens dreimaliger Ausführung sogar toleriert werden kann. Durch zusätzliche systematische Diversität kann sogar erreicht werden, dass sich die meisten permanenten Fehler wie temporäre Fehler auswirken und ebenfalls toleriert werden können. Virtuelle Duplexsysteme ermöglichen es somit, auf zusätzliche Hardware zu verzichten, indem der Zeitpunkt der Funktionserbringung als (in Grenzen) variabel betrachtet wird.

Entfernte Redundanz greift diese Idee auf, bezieht sie jedoch nicht auf die Zeit, sondern abstrahiert vom Ort der Ausführung. Durch die Verwendung signierter Nachrichten wird dieser Ort variabel, wodurch bei gemeinsamer Nutzung von Ressourcen ebenso zusätzliche Hardware entfallen kann. Im Gegensatz zu virtuellen Duplexsystemen „erkauft“ sich Entfernte Redundanz diesen Vorteil nicht durch zusätzliche Zeit, sondern durch zusätzliche Information bzw. Kommunikation. Welcher der beiden Aspekte schwerer wiegt, hängt in starker Weise vom jeweiligen Anwendungsgebiet ab, so dass eine allgemeingültige Aussage über die Vorteilhaftigkeit des einen oder des anderen Verfahrens kaum möglich erscheint. Im Unterschied zu virtuellen Duplexsystemen ist Entfernte Redundanz jedoch in jedem Fall nicht auf die Tolerierung temporärer Fehler eingeschränkt (oder permanenter Fehler, deren Auswirkung durch systematische Diversität temporär begrenzt wird).

Doppelduplexsysteme (Siewiorek & Swarz, 1982) sind ein typisches Beispiel dafür, wie fehlertolerante Systeme baukastenartig zu größeren Einheiten zusammengestellt werden können: Zwei für sich genommen nur fehlererkennende Duplexsysteme werden hier parallel betrieben. Ein einzelner Fehler in einem der beiden Systeme wirkt sich nun so aus, dass das entsprechende System keine (bzw. keine nicht als ungültig erkennbare) Ausgabe liefert. Somit ist sichergestellt, dass die zuerst eintreffende Ausgabe eines beliebigen der beiden Systeme stets gültig ist. Die durch Doppelduplexsysteme geschaffene Möglichkeit der Kombinierbarkeit bereits bestehender Teilsysteme ist im Sinne einer Komplexitätsreduzierung besonders attraktiv. Zudem kann auf eine Mehrheitsentscheidung, welche bei Indeterminismus problematisch sein kann, zugunsten von zwei Vergleichen verzichtet werden, da diese stets gültige Aussagen über Fehler liefern, wenn es gelungen ist, jedes der beiden Duplexsysteme für sich deterministisch zu realisieren.

Das Konzept der Entfernten Redundanz ist insofern hiermit verwandt, als es ebenso die Voraussetzungen dafür schafft, bestehende Systeme durch weitere Komponenten nach vorgegebenen Schemata zu ergänzen und dabei größere Teilsysteme mit besseren Fehlertoleranzeigenschaften zu erzeugen. Kapitel 4.3 dieser Arbeit zeigte ausgehend von einem

nicht-fehlertoleranten System verschiedene Stufen eines solchen Prozesses bis hin zum allgemeinen Fall eines n-von-m-Systems mit Entfernter Redundanz. Durch den Wegfall der Notwendigkeit einer physikalischen Überkreuzverkabelung sind dabei die Eingriffe in das bereits bestehende System so gering wie möglich. Entfernte Redundanz ist zugleich insofern flexibler als das Konzept der Doppelduplexsysteme, als es für beliebige Fehlertoleranzgrade eingesetzt werden kann. Ein weiterer Vorteil besteht darin, dass stets nur die tatsächlich benötigte strukturelle Redundanz verwendet wird, während Doppelduplexsysteme quasi verschwenderisch mit dieser Ressource umgehen, da zur Erzielung von Einfehlertoleranz vier (anstelle von prinzipiell drei notwendigen) Komponenten benötigt werden. Wie in der vorliegenden Arbeit gezeigt wurde, kommt Entfernte Redundanz bei Nutzung bereits bestehender Ressourcen sogar mit nur zwei Komponenten (ergänzt durch Prozesse auf einem entfernten Rechner) aus.

Gegenseitige Redundanz (Echtle, 1990) ist eine Variante dynamischer struktureller Redundanz, welche darauf abzielt, den Aufwand für Redundanz dadurch gering zu halten, dass Komponenten sich im Fehlerfall gegenseitig als Redundanz zur Verfügung stehen und die Aufgaben der jeweils anderen Komponente zusätzlich zu der eigenen mit erbringen. Dies setzt voraus, dass, sobald tatsächlich ein Fehler auftritt, genügend Kapazität für die Fehlerbehandlung vorhanden ist, schafft jedoch die Möglichkeit, diese Kapazität in der Zeit bis zum Auftreten des Fehlers für andere Zwecke zu nutzen.

Entfernte Redundanz hat es ebenso zum Ziel, ansonsten ungenutzte Kapazität nutzbar zu machen. Im Unterschied zu gegenseitiger Redundanz ist die Zuordnung in den Beispielen dieser Arbeit statischer Natur. Die für die Aufrechterhaltung der Funktion im Fehlerfall notwendige Kapazität muss jedoch bei beiden Konzepten vorhanden sein. Auch Entfernte Redundanz kann daher grundsätzlich zunächst auf das zur Fehlererkennung notwendige Minimum beschränkt bleiben, wobei die restliche Rechenzeit unkritischen Funktionen zugesprochen wird. Bei Auftreten eines Fehlers wären diese zugunsten der für den sicheren Betrieb des Systems notwendigen Funktionalität zu stornieren. Auf diese Weise kann Entfernte Redundanz die Merkmale gegenseitiger Redundanz nutzen und schafft durch den Wegfall direkter Kabelverbindungen zugleich die Möglichkeit, Funktionen innerhalb des Gesamtsystems auf eine potenziell größere Anzahl fremder Rechner zu verlagern. Vorauszusetzen ist dabei die Möglichkeit einer Koexistenz von sicherheitskritischen und nicht-sicherheitskritischen Funktionen, was mit den in Kapitel 4.2 beschriebenen Maßnahmen erreicht werden kann.

Der über die in den vorangegangenen Abschnitten aufgeführten Konzepte hinausgehende Beitrag Entfernter Redundanz kann somit insgesamt wie folgt zusammengefasst werden:

- Entfernte Redundanz ermöglicht es, den Ort der Funktionserbringung sicherheitskritischer Anwendungsprozesse unabhängig vom Ort der durch sie angesteuerten Peripherie zu wählen. Hierdurch wird eine wesentliche Voraussetzung dafür geschaffen, vorhandene Rechenkapazitäten in fehlertoleranten Steuerungen bestmöglich auszunutzen.
- Entfernte Redundanz erlaubt es, durch zusätzliche Buskommunikation die übrige Verkabelung der Komponenten auf ein Mindestmaß zu reduzieren. Es können hierdurch Systeme mit einem beliebigen Grad an Fehlertoleranz baukastenartig entworfen werden, wobei keine aus Fehlertoleranzsicht entbehrliche Hardware benötigt wird.
- Entfernte Redundanz gestattet es, ein 2-von-3-System mit 2 Rechnern, ein 3-von-5 System mit 3 Rechnern oder allgemein ein n-von-m-System mit nur n physikalisch tatsächlich vorhandenen Rechnern zu realisieren (vorausgesetzt, weitere Rechner, die primär für andere Zwecke vorgesehen sind, können mitbenutzt werden).

All dies ist möglich, ohne zugleich eine Verringerung der Fehlertoleranzeigenschaften vorauszusetzen. Noch stärker komprimiert könnte man also sagen:

> *Entfernte Redundanz ermöglicht es, den Spielraum für den Entwurf fehlertoleranter Steuerungssysteme hinsichtlich der Dimension „Ort der Funktionserbringung“ zu erweitern und schafft hierdurch die Voraussetzung, die im Gesamtsystem existierende strukturelle Redundanz in skalierbarer Art und Weise maximal auszunutzen, ohne dabei die durch die Anwendung vorgegebenen Fehlertoleranzeigenschaften zu mindern.*

6.2 Einfluss auf ökonomische Faktoren

Um einen Kostenvergleich zwischen Entfernter Redundanz und Dedizierter Redundanz anstellen zu können, ist zunächst zu klären, welche grundsätzlichen Faktoren dabei zu berücksichtigen sind. Betrachtet man Lebenszykluskostenmodelle (Zehbold, 2001), so wird ferner offenbar, dass im Sinne einer ganzheitlichen Beurteilung neben den bei der Herstellung eines einzelnen Systems anfallenden Kosten noch weitere Aspekte einbezogen werden sollten. Es sind dies zunächst die Entwicklungskosten (pro Stück), welche noch vor der eigentlichen Fertigung anfallen. Hinzu kommen durch den laufenden Betrieb entstehende Kosten sowie ggf. auch die Kosten für eine Stilllegung des Systems. Über den gesamten Lebenszyklus hinweg ergibt sich damit die in Abbildung 97 dargestellte Situation:

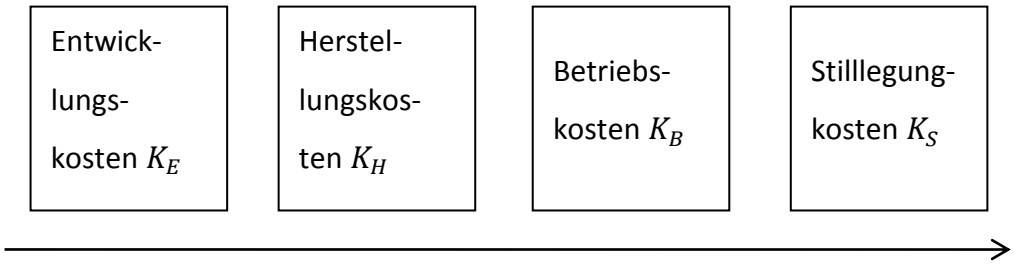


Abbildung 97: Vereinfachter Produktlebenszyklus

Unter Zugrundelegung dieses vereinfachten Produktlebenszyklus entstehen als Gesamtkosten:

$$K = K_E + K_H + K_B + K_S$$

Nicht alle genannten Größen lassen sich an dieser Stelle unmittelbar quantifizieren. Fokus der vorliegenden Arbeit ist es vielmehr, dort, wo es möglich ist, eine Tendenz aufzuzeigen und dort, wo die Einflussfaktoren klarer sind, selbige anzugeben. Zu beachten ist ferner, dass die in den verschiedenen Phasen anfallenden Kosten ggf. unterschiedlichen Trägern zuzurechnen sind. Da eine Untersuchung dieser Beziehungen detaillierte Informationen etwa über Geschäftsmodelle erfordert, soll dieser Aspekt hier nicht näher betrachtet werden.

Die Entwicklungskosten für Systeme mit Entfernter Redundanz sind zunächst sicherlich vergleichsweise hoch anzusetzen. Als Gründe hierfür sind zum einen neu zu ent-

wickelnde Hard- und Software zu nennen, zum anderen aber auch Kosten für den mit der Einführung einer Technologie notwendigen Wissenserwerb (Schulungen etc.) sowie ggf. anfallende Zertifizierungskosten. Hierbei ist zu berücksichtigen, dass in vielen Branchen eine für den Einsatz von Entfernter Redundanz geeignete Softwareinfrastruktur bereits vor der Einführung steht oder in Planung ist (vgl. Kapitel 6.3). Für den Serieneinsatz geeignete Hardware zur Signaturerzeugung und -prüfung wird jedoch i. d. R. zunächst zu entwickeln sein. Zu beachten ist dabei auch, dass ein für den jeweiligen Einsatzzweck geeignetes Signaturverfahren auszuwählen bzw. sogar erst zu entwerfen ist. Sobald jedoch diese Grundlagen geschaffen sind, ist anzunehmen, dass die Entwicklungskosten durch wachsende Erfahrung und aus Gründen der Wiederverwendbarkeit einmal geschaffener Lösungen zunehmend geringer werden und sich den Kosten für Dedizierte Redundanz annähern oder sie ggf. – bedingt durch die dem Konzept innewohnende Notwendigkeit einer gewissen Standardisierung – langfristig sogar unterschreiten. Die Zertifizierung von Systemen mit Entfernter Redundanz kann aufgrund einer stärkeren Verflechtung unterschiedlicher Teilsysteme (wie z. B. bei der Weiterleitung von Nachrichten durch fremde Knoten) und damit auch unterschiedlicher Verantwortungen zunächst ebenfalls aufwändiger werden. Kapitel 6.5 zeigt jedoch, dass sie dennoch möglich ist und wie sich aktuelle Normen dahingehend entwickeln, den Zertifizierungsprozess gerade unter diesen Bedingungen gezielt zu erleichtern.

In Bezug auf die Herstellungskosten ist zunächst von Interesse, wie sich die Verwendung Entfernter Redundanz auf die Anzahl der für ein System notwendigen Hardwarekomponenten auswirkt. Eine Aufstellung dieses Hardwareaufwandes im Vergleich zu Dedizierter Redundanz liefert nachfolgende Tabelle 14.

Wie bereits ausgeführt, erfordert Entfernte Redundanz grundsätzlich weniger Steuergeräte als Dedizierte Redundanz, sofern sich mehrere Regelkreise Rechenkapazitäten teilen können. Für das in dieser Arbeit vorgestellte Schema eines n-von-m-Systems mit Entfernter Redundanz sind nur n Knoten mit der Peripherie verbunden, so dass sich $m-n$ Knoten potentiell einsparen lassen. Es entfallen des Weiteren $m-n$ Positionssensoren, falls es die Anwendung erlaubt, die Position aus den Werten vorhandener Rotationssensoren zu berechnen. Die Anzahl der Rotationssensoren ist für beide Systemvarianten identisch $n \cdot (m-n)$, da in beiden Fällen dieselbe Anzahl an Aktuatoren zu überwachen sind und es nicht vorkommen darf, dass ein Aktuator und alle ihn überwachenden Rotationssensoren zugleich fehlerhaft sind. Auch die Anzahl der Endstufen liegt für beide Systemvarianten aus

demselben Grund identisch bei $n \cdot (m - n)$. Die Anzahl der benötigten Kabelverbindungen ist je nach Ansatz jedoch stark unterschiedlich: Bei Dedizierter Redundanz müssen m Positionssensoren mit je einer ECU verbunden werden, bei Entfernter Redundanz sind es nur n Sensoren (erster Term der Formel in Tabelle 14). Dedizierte Redundanz erfordert die Verbindung von $n \cdot (m - n)$ Endstufen mit jeweils $m - 1$ ECUs. Bei Entfernter Redundanz sind die entsprechenden $n \cdot (m - n)$ Endstufen in Serie mit nur einer ECU verbunden. Gleiches gilt für die Rotationssensoren: bei Verwendung Dedizierter Redundanz sind hier $n \cdot (m - n)$ Rotationssensoren mit jeweils $m - 1$ ECUs verbunden, Entfernte Redundanz hingegen erfordert für die $n \cdot (m - n)$ Rotationssensoren jeweils lediglich eine Verbindung zu einer ECU (zweiter Term der o. g. Formel). Identisch für beide Varianten ist die Verbindung von n Steuergeräten mit den ihnen zugeordneten Aktuatoren (dritter Term).

Tabelle 14: Hardwareaufwand Dedizierte/Entfernte Redundanz

	2-v-3-Syst.		**3-v-5-Syst.**		**4-v-7-Syst.**		**n-von-m-System**	
	R_{ded}	**R_{rem}**	**R_{ded}**	**R_{rem}**	**R_{ded}**	**R_{rem}**	**R_{ded}**	**R_{rem}**
ECUs	3	2	5	3	7	4	m	n
P-Sens.	3	2	5	3	7	4	m	n
R-Sens.	2	2	6	6	12	12	n(m-n)	n(m-n)
Endst.	2	2	6	6	12	12	n(m-n)	n(m-n)
Kabel	13	8	56	18	155	32	m+2n(m-n)(m-1)+n	n+2n(m-n)+n

Da ein Großteil der Kommunikation bei Verwendung Entfernter Redundanz nicht über direkte Kabelverbindungen, sondern über das Bussystem verläuft, erhöht sich, während der Hardwareaufwand sinkt, der diesbezügliche Kommunikationsoverhead im Vergleich zu Dedizierter Redundanz. Auch hier ist zwischen den verschiedenen Komponenten (bzw. den von ihnen ausgesandten Nachrichten) zu unterscheiden. So liegt die Anzahl der zu übertragenden Sollvorgaben für beide Systemvarianten identisch bei m Nachrichten, sofern eine fehlertolerante Übermittlung gewünscht ist. Die Werte der Rotationssensoren werden bei Verwendung Dedizierter Redundanz über direkte Kabelverbindungen übermittelt, so dass keine Buslast anfällt. Bei Entfernter Redundanz sind hingegen $n \cdot (m - n)$ Nachrichten erforderlich, was der Anzahl der Rotationssensoren entspricht. Die entsprechenden Werte sind zwar $m - 1$ Knoten zur Verfügung zustellen, dies kann jedoch bei Entfernter Redundanz per Multicast geschehen, während bei Dedizierter Redundanz diese Möglichkeit nicht besteht (und entsprechend mehr Kabelverbindungen benötigt werden). Auch für die Passivierung sind bei Verwendung Dedizierter Redundanz keine Busnachrichten notwendig. Bei Entfernter

Redundanz hingegen fallen $n \cdot (m-1)$ Nachrichten an, da jene n Knoten, welche über Aktuatoren verfügen, von $m-1$ fremden Knoten überwacht werden. Auch hier ist Multicast möglich, so dass der Faktor $m-n$ für die Anzahl der Endstufen pro Knoten im Gegensatz zum Aufwand für die Verkabelung bei Dedizierter Redundanz entfällt.

Eine Gegenüberstellung des Kommunikationsoverheads beider Varianten liefert nachfolgende Tabelle 15, wobei unterstellt wird, dass bei Verwendung Entfernter Redundanz eine näherungsweise Berechnung der Position alleine aus den Rotationsangaben möglich ist:

Tabelle 15: Erforderliche Buskommunikation Dedizierte/Entfernte Redundanz

	2-v-3-Syst.		**3-v-5-Syst.**		**4-v-7-Syst.**		**n-von-m-System**	
	R_{ded}	R_{rem}	R_{ded}	R_{rem}	R_{ded}	R_{rem}	R_{ded}	R_{rem}
Sollwert	3	3	5	5	7	7	m	m
Rotation	0	2	0	6	0	12	0	n(m-n)
Passiv.bef.	0	2	0	3	0	4	0	n(m-1)

Insgesamt zeigt sich demnach für den Kommunikationsoverhead bei Entfernter Redundanz ein quadratischer Zusammenhang bezogen auf die Anzahl der zu tolerierenden Fehler f $(= n - 1 = \left\lfloor \frac{m}{2} \right\rfloor)$. Da bei Nutzung Dedizierter Redundanz jedoch kein Multicast von Nachrichten möglich ist, findet sich dort sogar ein kubischer Zusammenhang zwischen der Anzahl der zu tolerierenden Fehler und der Anzahl der benötigten Verbindungskabel. Während bereits bei relativ kleinen Redundanzgraden ein solcher Aufwand kaum mehr wirtschaftlich zu handhaben sein dürfte (vgl. erneut Tabelle 14), sind die heute eingesetzten Bussysteme leistungsfähig genug, um die durch Entfernte Redundanz entstehenden Anforderungen in Bezug auf den Mehrbedarf an Kommunikation leicht zu erfüllen (s. Kapitel 6.3).

Bei alldem ist zu beachten, dass die Anzahl der von einem fehlertoleranten System zu tolerierenden Fehler typischerweise gering ist, so dass die meisten entsprechenden Systeme auf Einfehlertoleranz ausgelegt sind, während Doppel- oder Mehrfehlertoleranz eher bei speziellen Anwendungen zum Einsatz kommt, welche aus anderen Gründen ohnehin komplex und teuer sind. In diesen Bereichen dürften die Entwicklungskosten eine größere Rolle spielen als die Materialkosten bei der Herstellung. Allerdings lässt sich mit Blick auf solche Systeme die mit Entfernter Redundanz einhergehende Vereinfachung der Systemstruktur ggf. auch in anderen Phasen des Produktlebenszyklus kostensenkend nutzen. Ansonsten wird bezüglich des Hardwareaufwands die anzusetzende Stückzahl ein wichtiger Faktor sein. Gerade in Bereichen wie etwa der Automobilwirtschaft kann daher bereits der Wegfall einer einzelnen Komponente bedeutsam sein.

In Bezug auf die während des Betriebs anfallenden Kosten sind die o. g. Zusammenhänge zunächst insofern relevant, als sie in nicht unerheblichem Maß in das Gewicht des fertigen Systems einfließen dürften. Insbesondere im Luft- und Raumfahrtbereich ist dieser Faktor von entscheidender Bedeutung, entweder weil Nutzlast nur begrenzt zur Verfügung steht oder weil über eine Betriebsdauer von ca. 30 Jahren das Gewicht einen beträchtlichen Einfluss auf die Treibstoffkosten hat. Insgesamt ist hier Entfernte Redundanz durch den geringeren Bedarf an Hardwarekomponenten (einschl. Kabel) als vorteilhaft anzusehen. Auch im Zusammenhang mit Umweltschutzaspekten kann dieser Aspekt bedeutsam sein. Weitere Kosten während der Phase des Betriebs können durch Wartungen entstehen. Auch wenn diesbezüglich prinzipiell eine Vielzahl von Einflussfaktoren als relevant zu erachten sind, so kann Entfernte Redundanz zumindest durch die Eigenschaft, weniger Kabel und Steckverbindungen zu erfordern, als günstig angesehen werden: Da diese Komponenten als besonders fehleranfällig gelten, sind geringere Wartungskosten zu erwarten, sofern die zur Signaturerzeugung und -prüfung verwendete, zusätzlich erforderliche Hardware hinreichend zuverlässig ist.

Ein möglicher Einflussfaktor auf die Stilllegungskosten sind Art und Menge der verwendeten Hardwarekomponenten sowie die daraus resultierende Wiederverwendbarkeit von Systemteilen. Da sich das Konzept der Entfernten Redundanz jedoch gegenwärtig in einer frühen Entwurfsphase befindet, erscheint eine abschließende Einschätzung zum jetzigen Zeitpunkt unzulässig.

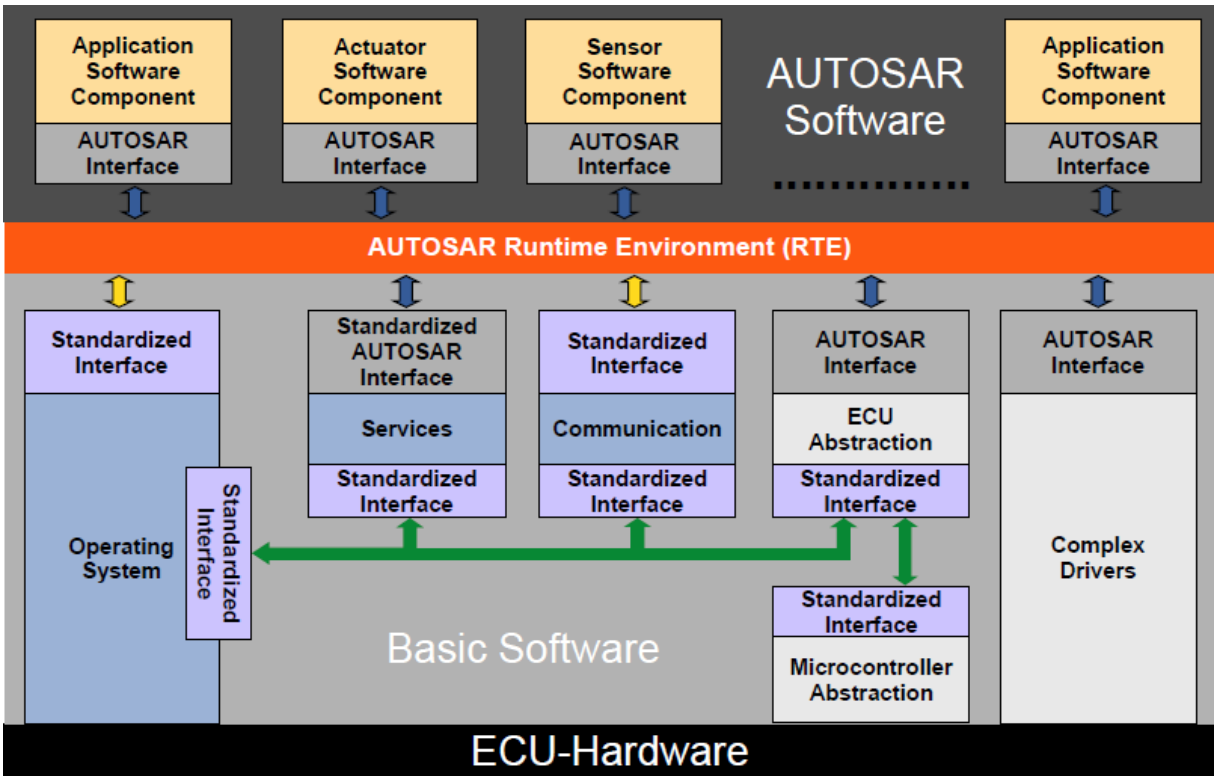
6.3 Bezug zu aktuellen technologischen Entwicklungen

Zusätzlich zum Einfluss auf ökonomische Faktoren erscheint es für eine Bewertung des Konzepts ebenso lohnend, die Vereinbarkeit Entfernter Redundanz mit aktuellen technologischen Entwicklungen zu untersuchen. Diese Entwicklungen sind insofern von Bedeutung, als sie die Rahmenbedingungen für den Einsatz Entfernter Redundanz darstellen und diesen als solche wesentlich erleichtern oder erschweren können. Auch wenn die Betrachtung im Rahmen der vorliegenden Arbeit insgesamt vergleichsweise allgemein bleiben muss, werden im Folgenden, auch um ggf. unterschiedlichen Bedingungen Rechnung tragen zu können, die Bereiche „Hardware“, „Software“, „Datenbusse“ und „Gesamtsystem“ separat betrachtet.

Bezüglich der in eingebetteten Systemen verwendeten Hardware kann beobachtet werden, dass grundsätzlich Steuergeräte mit immer leistungsfähigeren Prozessoren zur Verfügung stehen. Aufgrund der stark unterschiedlichen Anforderungen der einzelnen Teilsysteme, wie sie etwa durch Beleuchtungsanlage, Motorsteuerung oder „Infotainment“ in einem Automobil gegeben sind, existiert jedoch weiterhin eine sehr große Bandbreite unterschiedlich dimensionierter Mikrocontroller. Diese Tatsache schränkt zunächst die Möglichkeit der Verlagerung von Software auf fremde Steuergeräte im Rahmen Entfernter Redundanz ein. Zugleich sind jedoch aufgrund des steigenden Kostendrucks Konsolidierungsprozesse beobachtbar (vgl. die folgenden Abschnitte), welche zumindest für bestimmte Teilbereiche eine Vereinheitlichung der Hardware-/Softwarearchitektur erwarten lassen und so ihrerseits den Einsatz Entfernter Redundanz erleichtern könnten. Da sich somit jedoch insgesamt gegenläufige Entwicklungen gegenüberstehen, erscheint eine eindeutige Beurteilung der Situation derzeit unangemessen.

Der Aspekt der Software ist in mehreren für Entfernte Redundanz relevanten Branchen gegenwärtig im Wandel begriffen, wobei ein klarer Trend hin zu wiederverwendbaren Komponenten und damit zugleich weg von monolithischen Systemen zu beobachten ist. Das grundsätzliche Konzept besteht dabei meist darin, die Anwendungssoftware durch eine einheitliche Laufzeitumgebung möglichst unabhängig von der darunterliegenden Hardware zu halten und so ihre Wiederverwendbarkeit zu steigern. Darüber hinaus kann auf diese Weise ein großer Teil des Gesamtsystems im Rahmen strategischer Partnerschaften gemeinsam entwickelt werden, was eine weitere Möglichkeit zur Kostensenkung für alle Beteiligten darstellt.

Die in den letzten Jahren für den Automobilbereich geschaffene AUTOSAR-Architektur etwa (s. Abbildung 98) kann als typisches Beispiel für eine solche Entwicklung gelten. Durch eine einheitliche Laufzeitumgebung werden sämtliche Zugriffe auf Betriebssystem, Kommunikationsstack und Peripherie gekapselt. Des Weiteren wird in diesem Rahmen die – u. a. auch für Entfernte Redundanz relevante – Möglichkeit geschaffen, unter Berücksichtigung von unterschiedlichen Sicherheitsanforderungen bzgl. der Verwendung von Speicher und Laufzeit mehrere Anwendungen gemeinsam auf einem Steuergerät auszuführen (vgl. Kapitel 4.2).



(AUTOSAR, 2010)

Abbildung 98: AUTOSAR-Softwarearchitektur

Als Entsprechung im Luftfahrtbereich ist der Standard ARINC 653 anzusehen. Mit Blick auf branchenspezifizisch besonders strenge Auflagen können auch hier Anwendungen unterschiedlicher Sicherheitsansprüche isoliert voneinander und unter Berücksichtigung von Echtzeitanforderungen auf einem gemeinsamen Steuergerät zur Ausführung gebracht werden (Wind River Systems, 2008).

Beide genannten Beispiele verdeutlichen eine sehr ähnliche Gesamtentwicklung, wobei Entfernte Redundanz hiervon allerdings grundsätzlich unabhängig ist und sich bereits für einzelne Teilsysteme, etwa durch den Wegfall der für fehlertolerante Systeme ansonsten typischen Verkabelung „über Kreuz“, lohnen kann. Um jedoch die durch Entfernte Redundanz geschaffenen Möglichkeiten in vollem Maße auszunutzen, insbesondere also Steuergeräte von mehreren Anwendungen gemeinsam nutzen zu können und so die Anzahl der insgesamt benötigten ECUs zu senken, sind die genannten Entwicklungen im Bereich der Softwarearchitektur in hohem Maße geeignet.

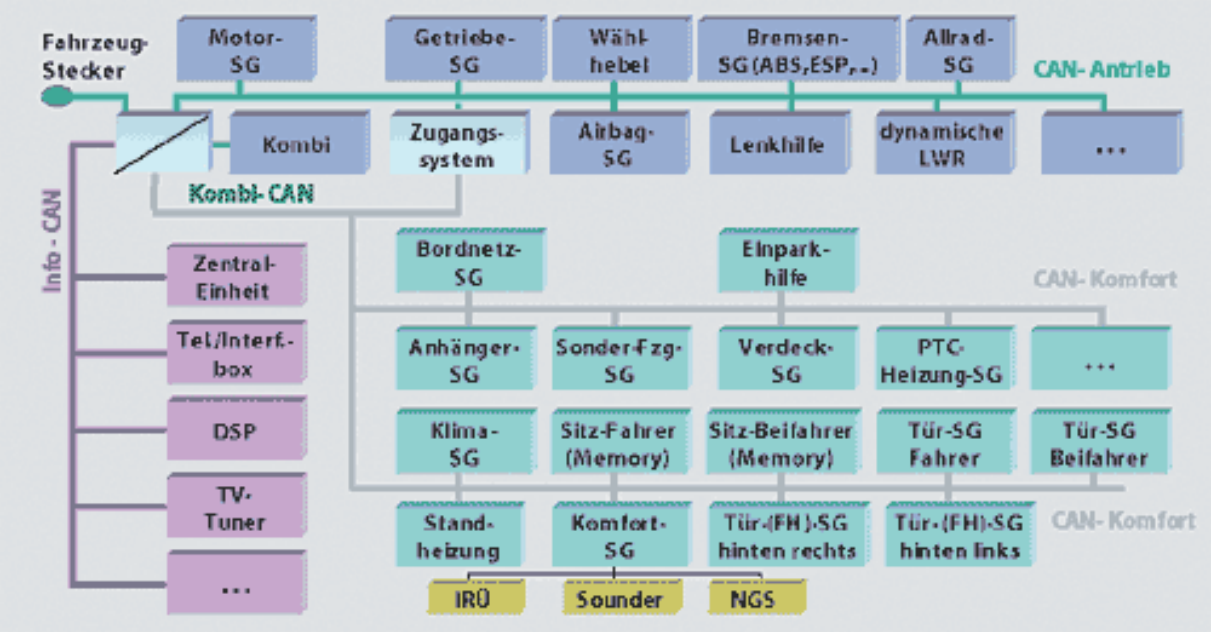
Das Vorhandensein leistungsfähiger Datenbussysteme ist für die Einsetzbarkeit Entfernter Redundanz von besonderer Bedeutung, da, wie bereits weiter oben beschrieben, das Konzept für den Wegfall direkter Kabelverbindungen zusätzliche Buskommunikation

erfordert. Je höher der gewünschte Grad an Fehlertoleranz, desto höher sind die so entstehenden Anforderungen an das Bussystem. Diese Anforderungen beziehen sich dabei vornehmlich auf die erreichbare Datenübertragungsrate, aber auch bereits in das Bussystem integrierte Fehlertoleranzverfahren können die Einführung Entfernter Redundanz erleichtern.

Grundsätzlich sind für alle durch Entfernte Redundanz berührten Branchen echtzeitfähige Busse vorhanden. Typische Vertreter sind etwa FlexRay für den Automobilbau, AFDX[21] in der Luftfahrt und TCN[22] im Eisenbahnbereich. Mit Übertragungsraten im Megabit-Bereich ist der durch Entfernte Redundanz entstehende zusätzliche Kommunikationsaufwand auch bei geforderten Zykluszeiten von etwa 2 - 10 ms je nach Anwendungsgebiet weiterhin zu erreichen (vgl. etwa die Situation im Beispielsystem einer elektronisch geregelten Lenkung aus Kapitel 5.2.3.2). Alle genannten Busse können zudem mit zwei redundanten Kanälen betrieben werden, was bereits zur Erreichung von Einfehlertoleranz notwendig ist. Auch nicht-fehlertolerante Bussysteme können allerdings durch Verwendung zusätzlicher Fehlertoleranzverfahren für Entfernte Redundanz eingesetzt werden, jedoch wäre der Aufwand dann natürlich entsprechend höher. So ist etwa bei mehr als zwei „entfernt redundanten"-Knoten sicherzustellen, dass beide eine einheitliche Sicht auf das System haben. Sofern das Bussystem dies nicht von sich aus garantiert, sind Übereinstimmungsprotokolle auf Anwendungsebene zu verwenden.

Bezogen auf das Gesamtsystem ist im Automobilbereich im Laufe der Jahre ein Nebeneinander zahlreicher unterschiedlicher Architekturen entstanden. Die Zahl der Steuergeräte ist dabei kontinuierlich gestiegen, um zusätzliche Funktionen zu verwirklichen, so dass über 50 Steuergeräte pro Automobil derzeit keine Seltenheit sind (Ebert & Jones, 2009). Gegenwärtig existieren daher Bestrebungen, mit einem leistungsfähigen Bus wie etwa FlexRay als „Backbone" eine Integration der verschiedenen Teilsysteme zu erleichtern. Zugleich wird versucht, den Herausforderungen immer neuer Varianten nicht nur durch gemeinsame Plattformen, sondern auch durch Standards wie etwa AUTOSAR zu begegnen. Neuere Anwendungen, welche z. B. das Lenkverhalten an die aktuelle Geschwindigkeit des Fahrzeugs anpassen, erfordern dabei sogar eine Kommunikation über die Grenzen bestehender Teilsysteme hinweg, so dass sich das Gesamtsystem typischerweise wie in Abbildung 99 gezeigt darstellt und damit eine wesentliche Voraussetzung Entfernter Redundanz erfüllt: die Vernetzung einer Vielzahl von Steuergeräten mittels geeigneter Bussysteme.

[21] Avionics Full Duplex Switched Ethernet
[22] Train Communication Network



(EASIS, 2005)

Abbildung 99: Beispiel eines modernen Fahrzeugnetzwerks

Im Luftfahrtbereich wurden in der Vergangenheit typischerweise monolithische Systeme verwendet. Auch hier sind es Kostenüberlegungen und steigende Anforderungen zugleich, welche dazu geführt haben, dass in neueren Flugzeugen (so etwa im Airbus A380) bislang eigenständige Systeme über einen gemeinsamen Bus (hier: AFDX) gekoppelt werden. Zusammen mit der durch ARINC 653 geschaffenen einheitlichen Softwarearchitektur wird von diesem als IMA (Integrated Modular Avionics) bezeichneten Konzept ein erhebliches Einsparpotential erwartet (Avionics Magazine, 2007).

Insgesamt kann demnach festgehalten werden, dass das Konzept der Entfernten Redundanz aktuellen technologischen Entwicklungen nicht etwa widerspricht, sondern in allen wesentlichen Punkten mit ihnen konform geht. Die erforderliche Hardware ist verfügbar, aktuelle Softwarearchitekturen unterstützen die gemeinsame Nutzung von Steuergeräten, leistungsfähige Busse erlauben es, den erforderlichen Kommunikationsoverhead zu bewältigen und die relevanten Branchen stellen sich auf die Nutzung der entstehenden Möglichkeiten in Form übergeordneter Konzepte ein oder haben sie bereits vollzogen. Entfernte Redundanz kann auf diesen Technologien aufsetzen und auf diese Weise die kostengünstige Realisierung fehlertoleranter Teilsysteme in einem solchen Verbund ermöglichen.

6.4 Auswirkungen auf den Systementwicklungsprozess

Die Verwendung Entfernter Redundanz hat potentiell Auswirkungen darauf, wie Systeme in der Praxis entwickelt werden. Anders als bei Dedizierter Redundanz sind beim Konzept der Entfernten Redundanz fremde Knoten als „Mitspieler“ eines Teilsystems beteiligt. Dies betrifft zum einen die signaturgeschützte Durchleitung von Nachrichten durch fremde Knoten, zum anderen die angestrebte gemeinsame Nutzung von Steuergeräten durch mehrere Anwendungen. Die möglichen Auswirkungen beider Szenarien sollen im Folgenden dargestellt werden.

Bei der Weiterleitung signaturgeschützer Nachrichten sind fremde Knoten notwendigerweise aktiv beteiligt. Zwar kann die Weiterleitung insofern transparent geschehen, als i. d. R. keine Modifikation von Nachrichten erforderlich ist. Dennoch sind Auswirkungen auf den gesamten Kommunikationsschedule zu berücksichtigen. Sollte bereits ein Schedule existieren, so sind ggf. Änderungen notwendig, wenn ein neu hinzukommendes Teilsystem den bzw. die betreffenden Knoten für die Weiterleitung von Nachrichten verwenden möchte.

Die gemeinsame Nutzung von Steuergeräten hat ebenfalls potenziell Auswirkungen auf fremde Knoten. Auch hier herrscht insoweit Transparenz, als die betroffenen Systeme nicht miteinander interagieren müssen. Bei der Verlagerung von Anwendungsteilen auf einen anderen Knoten ist jedoch zu prüfen, ob dort genügend Ressourcen (d. h. insbesondere Laufzeit und Speicher) zu Verfügung stehen.

Aktuelle Entwicklungen lassen erwarten, dass künftig eine einheitliche Softwarearchitektur existiert, welche die Verlagerung von Softwarekomponenten wesentlich vereinfacht (entsprechende Bestrebungen werden, wie bereits erwähnt, im Sinne einer besseren Wiederverwendbarkeit verfolgt). Ebenso kann davon ausgegangen werden, dass einzelne Teilsysteme über Datenbusse verfügen, welche ggf. durch Gateways miteinander verbunden werden können oder es bereits sind (EASIS, 2005). Die Realisierung der für Entfernte Redundanz erforderlichen Funktionalität reduziert sich damit auf ein Optimierungsproblem unter den Rahmenbedingungen, welche von den einzelnen Teilsystemen hinsichtlich

- verfügbarer Laufzeit,
- Echtzeitverhalten,
- Speicherbedarf und
- Zugriff auf den Kommunikationskanal

gegeben sind.

Im Zuge der in Kapitel 6.3 geschilderten Umstellung auf integrierte Gesamtsysteme in den einschlägigen Branchen stellen sich bereits ohne die Verwendung Entfernter Redundanz die oben genannten Probleme. Verfahren zu ihrer Bewältigung sind daher bereits in den entsprechenden Vorgehensmodellen enthalten, so dass Entfernte Redundanz insofern keine über das Bestehende hinausgehenden Ansprüche an den Entwicklungsprozess stellt. In einer für Entfernte Redundanz spezifischen Situation jedoch können zusätzliche Maßnahmen erforderlich werden: Wenn Ressourcen auch über Teilsystemgrenzen hinweg genutzt werden sollen, ist es notwendig, sich auf ein einheitliches Signaturverfahren zu einigen, das Verfahren zu implementieren und die entsprechenden Schlüssel (z. B. bei Systemerzeugung) zu verteilen. Eine solche Vorgehensweise kommt beispielsweise in (AUTOSAR, 2009) zum Einsatz und entspricht Forderungen, Sicherheitsaspekte als integralen Bestandteil des Entwicklungsprozesses zu betrachten (EASIS, 2005). Dies gilt umso mehr für Systeme, die ihrerseits zusätzliche Sicherheit aus einem Zusammenspiel mehrerer Teilsysteme bewirken sollen, wie etwa im Fall des sog. „adaptiven Kurvenlichts“, bei dem die Fahrzeugscheinwerfer in Abhängigkeit von Lenkwinkel und Geschwindigkeit geschwenkt werden können. Solche „Integrated Safety Systems“ sind bereits für sich genommen Faktoren, welche es erforderlich machen, o. g. Verfahren zu berücksichtigen.

Gelingt dies, so ergeben sich durch die Verwendung Entfernter Redundanz allerdings noch weitere Möglichkeiten: Auch im Bereich der Fehlertoleranz gibt es Bestrebungen, möglichst standardisierte, wiederverwendbare Entwurfsmuster zu etablieren (Auerswald, Herrmann, Kowalewski, & Schulte-Coerne, 2002). Erschwert wurde dieser Plan bislang unter anderem auch durch die Tatsache, dass Fehlertoleranz nicht etwa in Form einer zusätzlichen Komponente an ein bestehendes System angefügt werden kann, sondern strukturell an vielen Stellen in das System eingebunden sein muss. Entfernte Redundanz hebt diese Notwendigkeit in Teilen auf, da die Hardwareanbindung redundanter Komponenten durch dedizierte Kabel weitestgehend reduziert wird und die Softwareanbindung dieser Komponenten durch verfahrensbedingt einheitliche Schnittstellen gefördert wird. Dass somit auch fehlertoleranzbezogene Entwurfsmuster leichter realisiert werden können, ist insofern von Bedeutung, als damit in den betroffenen Branchen die Einführung einer durchgängigen Produktlinienentwicklung erleichtert wird.

6.5 Erfüllung rechtlicher Rahmenbedingungen

Rechtliche Rahmenbedingungen sind für die in dieser Arbeit behandelten Anwendungsbeispiele u. a. relevant bei der Erteilung einer Betriebserlaubnis durch die zuständigen Behörden. Rechtsvorschriften regeln in diesem Zusammenhang die übergeordneten Verfahren, wie etwa das Typgenehmigungsverfahren (EU-Richtlinie 70/156/EEC), definieren aber ebenso konkrete Vorgaben etwa zur Konstruktion von Lenksystemen in Kraftfahrzeugen. Darüber hinaus können die zulassenden Behörden zur Konkretisierung von Anforderungen auf Normen verweisen und ihre Einhaltung somit für bindend erklären.

Unabhängig hiervon schließt in Deutschland das Produkthaftungsgesetz eine Haftung bei durch Produktfehler entstandenen Sach- oder Personenschäden aus, wenn im Schadensfall der Nachweis gelingt, dass „der Fehler nach dem Stand der Wissenschaft und Technik in dem Zeitpunkt, in dem der Hersteller das Produkt in den Verkehr brachte, nicht erkannt werden konnte“ (§1 Abs. 2 Nr. 5). Um nun bestimmen zu können, was als Stand der Wissenschaft und Technik anzusehen ist, werden in der Praxis ebenfalls Normen herangezogen, wobei allerdings die Normentsprechung typischerweise lediglich als notwendig, aber nicht hinreichend gilt (Sauler & Kriso, 2009).

Als branchenübergreifende internationale Norm für die Entwicklung sicherheitsbezogener Systeme ist insbesondere die in Deutschland als DIN EN 61508 übernommene IEC 61508 von Bedeutung. Für zahlreiche Industriezweige existieren, teilweise hieraus abgeleitet, anwendungsspezifische Normen, so etwa ISO 26262 für Kraftfahrzeuge, EN 50128 für die Eisenbahnsteuerung und -überwachung, IEC 62061 zur Sicherheit von Maschinen und DO-178B in Verbindung mit Software im Luftfahrtbereich.

Grundkonzept all dieser Normen sind diskrete Sicherheitsanforderungsstufen (in IEC 61508 als „Safety Integrity Level“ / SIL bezeichnet), welche möglichen Fehlfunktionen ausgehend von der angenommenen Gefährdungshäufigkeit, der Beherrschbarkeit ihrer Auswirkungen und dem potentiellen Schaden zugeordnet werden (Schwarz, 2005). Auf Grundlage der Einstufung in ein bestimmtes SIL-Level werden unterschiedliche Anforderungen an den Hardware- bzw. Softwareentwicklungsprozess, aber auch an die Eigenschaften (etwa Ausfallraten) entsprechender (Teil-)Systeme gestellt.

Die meisten durch gesetzliche Regelungen berührten Merkmale der in dieser Arbeit beschriebenen Systeme betreffen nicht die gewählte Redundanzform (d. h. Dedizierte oder Entfernte Redundanz). So ist gegenwärtig die Genehmigungsfähigkeit von Systemen mit einer allein auf elektrischen Signalen basierenden Steuerung („X-by-Wire“) typischerweise vom

Anwendungsgebiet abhängig. Im Folgenden soll daher ausschließlich auf die Bedeutung der für Entfernte Redundanz spezifischen Eigenschaften eingegangen werden. Es sind dies:

- die gemeinsame Nutzung von Steuergeräten durch unterschiedliche Anwendungen (optional) sowie
- die Weiterleitung von Nachrichten über mindestens ein Steuergerät hinweg (erforderlich).

Am Beispiel von AUTOSAR lässt sich (ohne dass dies eine anwendungsbezogene Einschränkung darstellt) aufzeigen, wie eine erfolgreiche Zertifizierung in Bezug auf diese Merkmale erreicht werden kann:

Die gemeinsame Nutzung von Steuergeräten durch unterschiedliche Anwendungen ist kein zwingendes Merkmal von Entfernter Redundanz, bietet sich aber an, wenn hierdurch die Menge an Steuergeräten im Gesamtsystem reduziert werden kann. Entfernte Redundanz schafft hierfür insoweit die Voraussetzungen, als die physikalische Anbindung einiger Steuergeräte an die Peripherie überflüssig wird. Sofern an anderer Stelle im Netzwerk freie Rechenkapazitäten vorhanden sind, kann ihre Funktion dorthin verlagert werden. Dann ist allerdings nachzuweisen, dass sich mehrere Anwendungen (möglicherweise unterschiedlicher Sicherheitsstufen) auf ein und demselben Steuergerät nicht in schädlicher Weise beeinflussen können. Der AUTOSAR-Standard enthält bereits die hierfür notwendigen Mechanismen: Speicherschutz und Laufzeitüberwachung. Sofern diese beiden Module nach dem erforderlichen ASIL[23]-Level zertifiziert sind, dürfen gemäß ISO 26262 im Zuge einer sog. „ASIL-Dekomposition“ Anwendungen desselben ASIL-Levels neben beliebigen anderen Anwendungen und dem Laufzeitsystem gemeinsam auf einem Steuergerät verwendet werden (Wenzel, Fassl, & Kalmbach, 2010).

Die fehlertolerante Durchleitung von Nachrichten durch fremde Knoten erfordert eine geeignete Ende-zu-Ende-Absicherung der stattfindenden Kommunikation. Der AUTOSAR-Standard beinhaltet hierfür eine Bibliothek mit entsprechenden Mechanismen, welche je nach Anforderungsstufe zu „Profilen“ zusammengestellt werden können. Für die hier relevante Sicherstellung von Authentizität kommen neben Sequenznummern sog. „Data IDs“ als (vorab festgelegte) eindeutige Kennzeichen von Nachrichten zum Einsatz, welche in die CRC-Berechnung stets mit eingeschlossen und so vom Empfänger überprüft werden können. Je nach ASIL-Level hat der Anwender ggf. sicherzustellen, dass unterschiedliche Data IDs nicht den-

[23] Automotive SIL, eine branchenspezifische Adaption der SIL-Level aus IEC 61508

selben CRC aufweisen (AUTOSAR, 2009). Da das Konzept der Entfernten Redundanz die Wahl des Signaturverfahrens völlig freistellt, kann zur Erreichung des entsprechenden ASIL-Levels diese Anforderung entweder übernommen oder aber durch eine stärkere ersetzt werden.

Insgesamt kann demnach festgehalten werden, dass die für Entfernte Redundanz spezifischen Aspekte der Hard- und Softwarearchitektur problemlos nach den derzeit relevanten Normen zertifiziert werden können. Insoweit Rechtsvorschriften den Einsatz von X-by-Wire-Technologie für ein gegebenes System nicht per se ausschließen, wird also die Verwendung Entfernter Redundanz als genehmigungsfähig betrachtet werden können.

7 Fazit und Ausblick

7.1 Zusammenfassung der Ergebnisse

In der vorliegenden Arbeit wurde nach einer Erläuterung ausgewählter Begriffe zur Formulierung von Fehlertoleranzanforderungen und einer Einführung in den Aspekt der Redundanz als Grundlage der Fehlertoleranz zunächst das Konzept der Entfernten Redundanz vorgestellt. Dieses Konzept erlaubt es, mit Hilfe digitaler Signaturen fehlertolerante Systeme so zu konstruieren, dass auf bislang übliche Verkabelungsstrukturen „über Kreuz“ vollständig verzichtet werden kann. Da sendende Komponenten ihre Daten mit einer Signatur schützen und empfangende Komponenten diese Signatur prüfen, ist es stattdessen möglich, Daten über fremde Knoten und (i. d. R. bereits existierende) Bussysteme an ihr Ziel weiterzuleiten, ohne dass zwischenzeitliche Verfälschungen unentdeckt blieben.

Im weiteren Verlauf der Arbeit wurde gezeigt, wie aus einem nicht fehlertoleranten System mit Hilfe Entfernter Redundanz schrittweise ein fehlertolerantes System erzeugt werden kann. Ausgehend von der einfachsten Form, einem fehlererkennenden System, wurden dabei fehlertolerante Systeme zur Tolerierung von einem Fehler (2-von-3-System) und zwei Fehlern (3-von-5-System) gezeigt, bis hin zum allgemeinem Fall einer Tolerierung von k Fehlern in Form eines n-von-m-Systems (mit $n = k + 1$ und $m = 2 \cdot k + 1$). Für jedes der Systeme wurde dargelegt, warum die geforderte Fehlertoleranz erreicht wird. Für die beiden kleineren Systeme erfolgte zudem eine Gegenüberstellung mit äquivalenten Systemarchitekturen, welche unter Verwendung Dedizierter Redundanz entstanden. Das allgemeine Schema Entfernter Redundanz wurde als verifizierbarer Algorithmus formuliert, ferner wurden ein Referenzverfahren zur Signaturerzeugung und -prüfung sowie Möglichkeiten zur Effizienz- und Nutzensteigerung vorgestellt.

Im zweiten Teil der Arbeit wurde anhand des durchgehenden Beispiels einer elektronisch geregelten Lenkung gezeigt, wie Systeme mit Entfernter Redundanz in der Praxis realisiert werden können. Auf unterschiedlichen Abstraktionsebenen wurde der Entwurf zugleich einer gründlichen Analyse unterzogen: Zunächst wurde anhand einer Fehlerbaumanalyse nachgewiesen, dass die für Entfernte Redundanz notwendige Architektur hinsichtlich ihrer qualitativen und quantitativen Fehlertoleranzeigenschaften der Architektur mit Dedizierter Redundanz nicht unterlegen ist. Sodann wurde die Funktion der Lenkung in noch relativ stark abstrahierter Form als UPPAAL-Modell nachgebildet. Mittels formaler Verifikation konnte gezeigt werden, dass die Lenkung unter allen als möglich angenommenen Fehlfunktionen ihre Nutzfunktion weiterhin erbringen kann und die erforderlichen Maßnahmen zur Fehlerbe-

handlung stets eingeleitet werden. Der nächste Arbeitsschritt betraf die Erstellung einer detaillierteren funktionalen Simulation mit dem Programmpaket MATLAB/Simulink. Im Gegensatz zum vorhergehenden Modell konnte nunmehr die Lenkung mit ihren physikalischen Komponenten (u. a. den Stellmotoren) realistisch nachgebildet werden. Erstmals wurden die gesamte Buskommunikation sowie die Verfahren zur Signaturerzeugung und -prüfung explizit mitmodelliert. Auf diese Weise konnte gezeigt werden, dass das betrachtete System auch unter wirklichkeitsnahen Bedingungen realisierbar ist. Anhand von Fehlerinjektionsexperimenten wurde zudem erneut demonstriert, dass die implementierten Fehlertoleranzverfahren die Funktionsfähigkeit der Lenkung im Fehlerfall sicherstellen. Schließlich wurde mittels einer VHDL-Implementierung die technische Realisierbarkeit von Signaturerzeugung und -prüfung nach dem im ersten Teil der Arbeit vorgestellten Referenzverfahren nachgewiesen. Mit Hilfe einer Schaltungssimulation wurde zunächst die Fehlererkennungsrate analysiert. Diese wird durch den zugrundeliegenden CRC bestimmt, wobei die Wahl der zur Verschlüsselung verwendeten Faktoren entscheidend dafür ist, dass das Gesamtverfahren nicht schlechter als dieser abschneidet. Zu Demonstrations- und Lehrzwecken entstand eine FPGA-Umsetzung mit einer einfachen Anwendung, so dass gezeigt werden konnte, dass die für Entfernte Redundanz erforderlichen Komponenten unter den entsprechenden Rahmenbedingungen realisierbar sind, sich somit also auch als anwendungsspezifische integrierte Schaltung (ASIC) umsetzen lassen. Abschließend wurde mit Hilfe von zwei FPGA-Boards und zwei FlexRay-Knoten die gesamte Strecke von einem Sensor über zwei weiterleitende Knoten und das Bussystem hin zu einem Aktuator erfolgreich nachempfunden. Hierbei wurde ein Verfahren zur Zerlegung von Sequenznummern implementiert, welches es erlaubt, auf effiziente Weise das fehlerbedingte oder mutwillige wiederholte Senden gültig signierter, aber veralteter Daten zu erkennen.

Als Gesamtergebnis kann folglich festgehalten werden: Entfernte Redundanz ist in der Praxis realisierbar und leistet genau so viel wie Dedizierte Redundanz, sofern nicht:

- mehr Fehler auftreten, als im Fehlermodell vorgesehen (z. B. Doppelfehler bei einem 2-von-3-System)
- nicht tolerierbare Fehler auftreten, wie etwa das Brechen von Signaturen oder sog. „Common Cause-Fehler“

Es sei angemerkt, dass beide Einschränkungen selbstverständlich für alle Fehlertoleranzverfahren gelten.

Im dritten Teil der Arbeit wurde aus unterschiedlichen Blickwinkeln eine Bewertung des Konzepts der Entfernten Redundanz vorgenommen. Zunächst wurde der Beitrag Entfernter Redundanz für die Fehlertoleranzforschung aufgezeigt. Dieser liegt in der Schaffung eines zusätzlichen Freiheitsgrades bei der Erstellung fehlertoleranter Systeme, welcher es ermöglicht, die vorhandene Redundanz bestmöglich auszunutzen. Bei entsprechend vorhandenen Kapazitäten kann ein 2-von-3-System mit 2, ein 3-von-5-System mit 3 oder allgemein ein n-von-m-System mit nur n physikalischen Knoten realisiert werden. Anschließend wurde der Einfluss der Verwendung Entfernter Redundanz auf die über den Lebenszyklus eines Systems hinweg anfallenden Kosten untersucht. Hier gilt, dass die Entwicklungskosten zunächst höher anzusetzen sind. In Bezug auf die Herstellungskosten kommt jedoch der reduzierte Hardwareaufwand Entfernter Redundanz zum Tragen, wobei in der Praxis nicht nur die geringere Anzahl an Steuergeräten, sondern insbesondere der Wegfall der bislang notwendigen Verkabelung „über Kreuz“ von Bedeutung sein dürfte. Dieser wird erkauft durch zusätzliche Buskommunikation, was jedoch aus technischer Sicht handhabbar ist. Während der Nutzung von Systemen mit Entfernter Redundanz ist zunächst ein Gewichtsvorteil anzuführen (z. B. in der Luft- und Raumfahrt), insbesondere aber auch die bessere Wartbarkeit und die erhöhte Zuverlässigkeit durch den Wegfall von Steckverbindungen. In Bezug auf aktuelle technische Entwicklungen ist festzuhalten, dass diese eine ideale Voraussetzung für Entfernte Redundanz darstellen, da sie in vielen Fällen eine einheitliche und Hard- und Softwarearchitektur zum Ziel haben. Im Zusammenhang dieser Bemühungen ist auch der durch Entfernte Redundanz entstehende Zusatzaufwand beim Systementwurf zu bewerten. In Bezug auf die rechtlichen Rahmenbedingungen wurde schließlich aufgezeigt, dass die in der Praxis vieler Branchen notwendige Zertifizierung von Systemen mit Entfernter Redundanz problemlos gelingen kann.

7.2 Ausblick auf offene Forschungsfragen

In vielerlei Hinsicht erscheint die Erstellung eines realen Systems mit Entfernter Redundanz als ein lohnendes Ziel, durch das weitere Untersuchungen erst möglich werden. Dies betrifft zum einen die Fertigung signaturfähiger Komponenten (z. B. AD-/DA-Wandler), zum anderen könnte ein solches System zugleich als verbesserter Demonstrator dienen und die Idee Entfernter Redundanz unter Einsatzbedingungen und auf anschauliche Weise zeigen. Möglicherweise gelingt dies in Zusammenarbeit mit Kooperationspartnern aus der Industrie, welche das Konzept im Rahmen von Diskussionen und Tagungen bereits kennengelernt haben.

Da die Busauslastung durch Entfernte Redundanz steigt, bietet sich eine diesbezügliche Optimierung als weitere Fragestellung an. Es wird von der Anwendung abhängen, ob z. B. die Kosten für zusätzliche Sensoren höher sind, als die Kosten einer höheren Auslastung des Bussystems. Gerade dort, wo verschiedene Teilsysteme mit Entfernter Redundanz gemeinsam in einem Verbund zum Einsatz kommen, entsteht jedoch ein erheblicher Gestaltungsspielraum, insbesondere in Bezug auf dann ggf. notwendige Übereinstimmungsprotokolle.

Die Frage nach effizienten Signaturverfahren zeigt ebenfalls viele Möglichkeiten für weitere Optimierungen auf. Entfernte Redundanz schreibt kein bestimmtes Signaturverfahren vor. Je nach den Anforderungen der Anwendung könnte somit aus einer ganzen Bandbreite an Verfahren gewählt werden, von einfachen, aber möglicherweise weniger leistungsfähigen bis hin zu kryptografisch starken Verfahren. Auch kann untersucht werden, inwiefern die Verwendung von Kosignaturen zum Nachvollziehen der Weiterleitungskette einer Nachricht einen zusätzlichen Nutzen darstellen würden.

Ein weiterer Untersuchungsgegenstand könnten die Eigenschaften und Anforderungen unterschiedlicher Sensoren und Aktuatoren sein. Grundsätzlich existieren auch hier zunächst keinerlei Vorgaben. Ein typisches „Entwurfsmuster“ Entfernter Redundanz scheint jedoch darin zu bestehen, die zur Ansteuerung relevanten Größen sowohl absolut als auch relativ (d. h. in Bezug zu vergangenen Zeitpunkten) zu messen und gegebene Systemeigenschaften für Plausibilitätstests zu nutzen, um die insgesamt erforderliche Anzahl an Sensoren zu senken. Dies ist aber nicht zwingend notwendig, wie das in der Arbeit aufgeführte Beispiel elektronisch geregelter Landeklappen aufgezeigt hat. Als Aktuatoren sind neben Motoren auch Pumpen und Ventile, Propeller und Turbinen denkbar, prinzipiell also alle Arten von Aktuatoren, die elektronisch angesteuert werden können. Als Sensoren kommen von Rotations- und Positionssensoren über Geschwindigkeits- und Lagesensoren bis hin zu Sensoren zur Messung von Schalldruck oder Lichteinstrahlung ebenso zahlreiche Möglichkeiten in Betracht.

In der Tat könnten dadurch, dass zunächst weitere Beispiele auf das bestehende Verfahren übertragen werden, zusätzliche Anwendungsfelder Entfernter Redundanz erschlossen werden, welche ihrerseits neue Forschungsfragen aufwerfen. Die entstehenden Problemstellungen sind auch hier vielfältig: so wurde in (Echtle & Kimmeskamp, 2010) untersucht, wie das in der vorliegenden Arbeit untersuchte System weiterhin ohne Überkreuzver-

kabelung konstruiert werden kann, wenn eine aktive Bremse erforderlich ist, um beispielsweise nicht korrekt angesteuerte Eingangswellen eines Differentials arretieren zu können (s. Abbildung 100).

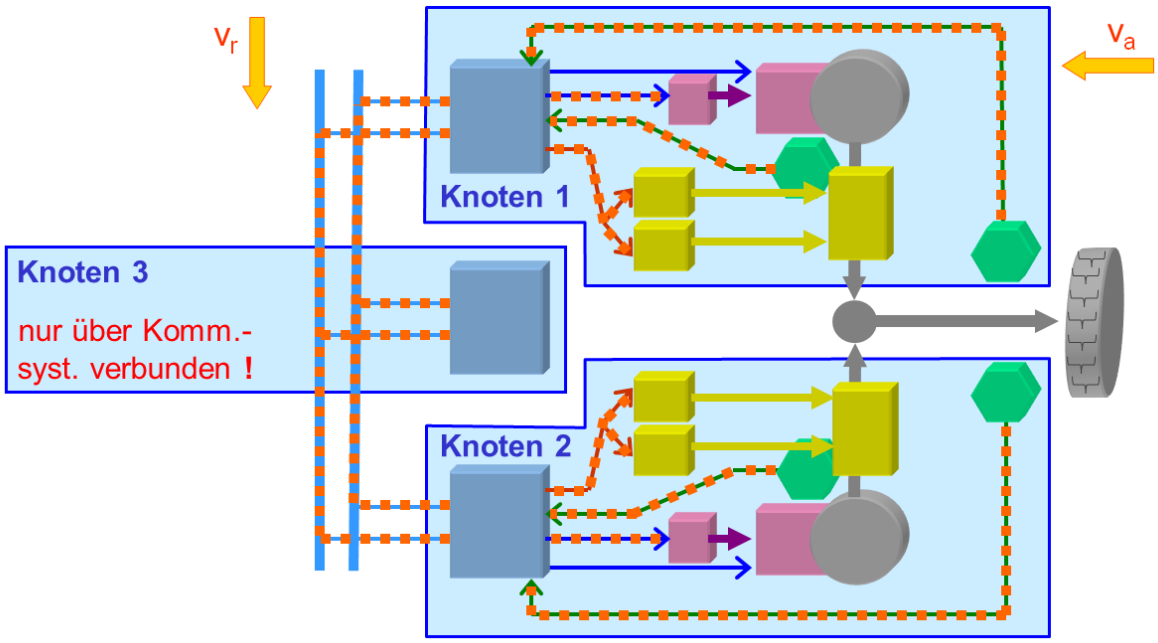


Abbildung 100: Variante mit aktiver Bremse

Die Verfügbarkeit solcher wiederverwendbarer Entwurfsmuster erscheint gerade auch für die Anwender von Fehlertoleranzverfahren wünschenswert. Bislang war es aufgrund der vielfältigen Schnittstellen entsprechender Verfahren mit dem Gesamtsystem schwierig, diesen Wunsch in die Tat umzusetzen. Es hat sich jedoch im Verlauf der Entstehung dieser Arbeit gezeigt, dass Entfernte Redundanz nicht nur klarere Schnittstellen schafft, sondern auch die Auseinandersetzung mit dem zu entwerfenden System intensiviert. Entfernte Redundanz verändert in vielen Fällen ebenso die Art und Weise, bereits bestehende Systeme zu analysieren (da Ort der Funktionserbringung und Funktion selbst unabhängig voneinander betrachtet werden). Oftmals wird hierdurch ein längst bestehendes Einsparungspotenzial sichtbar, welches lediglich noch nicht berücksichtigt wurde. Durch die Übertragung auf unterschiedliche Anwendungen könnte nun ein Katalog typischer Strukturen fehlertoleranter Systeme und ihrer Implementierung mit Entfernter Redundanz entstehen. Dies wäre sicherlich ein Gewinn für die Praxis, letztlich aber auch für die Fehlertoleranzforschung lohnend, da hierdurch weitere Ideen für auf den ersten Blick fehlerhaft erscheinende, tatsächlich aber lediglich unkonventionelle Lösungen gewonnen werden könnten.

Literaturverzeichnis

Altera. (4. Mai 2006). DE2 Development and Education Board: User Manual.

Angermann, A., Beuschel, M., Rau, M., & Wohlfarth, U. (2007). *MATLAB - Simulink - Stateflow: Grundlagen, Toolboxen, Beispiele.* München: Oldenbourg.

Auerswald, M., Herrmann, M., Kowalewski, S., & Schulte-Coerne, V. (2002). Entwurfsmuster für fehlertolerante softwareintensive Systeme. *at-Automatisierungstechnik, 50*(8), S. 389-398.

AUTOSAR. (17. Dezember 2009). *Specification of SW-C End-to-End Communication Protection Library.* Abgerufen am 31. März 2011 von http://www.autosar.org/download/R4.0/AUTOSAR_SWS_E2ELibrary.pdf

AUTOSAR. (24. November 2010). *Layered Software Architecture.* Abgerufen am 31. März 2011 von http://www.autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

Avionics Magazine. (2007). Integrated Modular Avionics: Less is More.

Avižienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing, 1*(1), S. 11-33.

Behrmann, G., David, A., & Larsen, K. G. (17. November 2004). *A Tutorial on Uppaal.* Abgerufen am 31. März 2011 von http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

David, A., Amnel, T., Stigge, M., & Ekberg, P. (16. November 2009). *UPPAAL 4.0 : Small Tutorial.* Abgerufen am 31. März 2011 von http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf

EASIS. (2005). *Deliverable D0.1.2: State of the art.* o. O.: The EASIS Consortium (Electronic Architecture and Systems Engineering for Integrated Safety Systems).

EASIS. (2006). *Deliverable D1.2-5: Discussions and findings on fault tolerance.* o. O.: The EASIS Consortium (Electronic Architecture and Systems Engineering for Integrated Safety Systems).

Ebert, C., & Jones, C. (2009). Embedded Software: Facts, Figures, and Future.

Echtle, K. (1987). Fault Tolerance based on Time-Staggered Redundancy. In F. Belli, & W. Görke (Hrsg.), *Fehlertolerierende Rechensysteme / Fault-Tolerant Computing*

Systems, 3. Internationale GI/ITG/GMA-Fachtagung (Tagungsband) (S. 348-361). London, UK: Springer.

Echtle, K. (1989). Distance Agreement Protocols. *Fault-Tolerant Computing Symposium (FTCS-19)* (S. 191-198). Los Alamitos, CA, USA: IEEE Press.

Echtle, K. (1990). *Fehlertoleranzverfahren.* Berlin, Heidelberg: Springer.

Echtle, K. (2003). Fehlertolerante verteilte Systeme. *(Vorlesungsmaterial).* Universität Duisburg-Essen.

Echtle, K. (2008). Zuverlässigkeit von Hardware und Software. *(Vorlesungsmaterial).* Universität Duisburg-Essen.

Echtle, K., & Kimmeskamp, T. (2009). Fault-Tolerant and Fail-Safe Control Systems Using Remote Redundancy. In K.-E. Großpietsch, A. Herkersdorf, S. Uhrig, T. Ungerer, & J. Hähner (Hrsg.), *22nd International Conference on Architecture of Computing Systems 2009 (ARCS '09): Workshop Proceedings* (S. 101-106). Berlin, Offenbach: VDE Verlag.

Echtle, K., & Kimmeskamp, T. (2010). Verification of a Control System built using Remote Redundancy by means of Timed Automata and State Space Exploration. In B. Müller-Clostermann, K. Echtle, & E. P. Rathgeb (Hrsg.), *15th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB & DFT 2010): Proceedings* (S. 8-23). Berlin, Heidelberg: Springer.

Echtle, K., & Tappe, D. (2006). *Project FlexBeam: Report of Work.* Universität Duisburg-Essen.

Echtle, K., Kimmeskamp, T., Jacquet, S., Malassé, O., Pock, M., & Walter, M. (2009). Reliability Analysis of a Control System Built Using Remote Redundancy. In L. Bartlett (Hrsg.), *Proceedings of the 18th Advances in Risk and Reliability Technology Symposium (AR2TS '09)* (S. 335-346). Loughborough, UK: Loughborough University.

Echtle, K., Kimmeskamp, T., Jacquet, S., Malassé, O., Pock, M., & Walter, M. (2010). Achieving Safety and Reliability for Systems with Remote Redundancy. (D. M. Clarke, R. Denning, D. Smith, & J. Christodoulou, Hrsg.) *The Journal of the Safety and Reliability Society, 30*(4), S. 36-55.

Formal Software Construction. (10. Februar 2005). OpenFTA: User Manual.

Friedrichs, B. (1995). *Kanalcodierung: Grundlagen und Anwendungen in modernen Kommunikationssystemen.* (H. Marko, & J. Hagenauer, Hrsg.) Berlin, Heidelberg: Springer.

Gajski, D. D., & Kuhn, R. (1983). Guest Editors Introduction - New VLSI Tools. *Computer, 16*(12), S. 11-14.

Gschwind, H. W., & Uebel, H. (1984). Fail-safe-Systeme mit redundanten Rechnern.

Haasl, D. F., Roberts, N. H., Vesely, W. E., & Goldberg, F. F. (1981). *Fault Tree Handbook.* Washington, D.C., USA: U.S. Nuclear Regulatory Commission.

Imig, M. (2009). Vergleichende Zuverlässigkeitsanalyse verschiedener Varianten eines Steer-by-Wire-Systems unter Berücksichtigung eines neuartigen Redundanzkonzeptes. *(Projektseminararbeit).* Universität Duisburg-Essen.

Jochim, M. (2003). *Automatische Erzeugung und Bewertung virtueller Duplexsysteme zur Erkennung von Betriebsfehlern in Mikroprozessoren.* Berlin: Logos.

Jochim, M. (2007). Zeitig steuern - Sichere Datenübertragung im Automobil. *c't Magazin für Computertechnik, 24*(2), S. 190-195.

Kimmeskamp, T. (2005). Formulierung von Fehlertoleranzanforderungen. In J.-T. Czornack, D. Stodden, C. Trinitis, & M. Walter (Hrsg.), *Diskussionskreis Fehlertoleranz 2005: Tagungsband* (S. 61-68). Aachen: Shaker Verlag.

Klenke, R. (17. Juni 1999). *Tutorial for VHDL Simulation with ModelSim and QSPro*. Abgerufen am 31. März 2011 von http://www.people.vcu.edu/~rhklenke/tutorials/vhdl/modules/m10_23/index.htm

Lala, P. K. (1985). *Fault tolerant and fault testable hardware design.* Englewood Cliffs, NJ, USA: Prentice Hall.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. (ACM, Hrsg.) *ACM Transactions on Programming Languages and Systems, 3*(4), S. 382-401.

Lang, H. W. (29. März 2011). *Codierungstheorie: CRC-Verfahren*. Abgerufen am 31. März 2011 von http://www.inf.fh-flensburg.de/lang/algorithmen/code/crc/crc.htm

Laprie, J.-C. (1985). Dependable Computing and Fault Tolerance: Concepts and Termiology. *15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15)* (S. 2-11). Los Alamitos, CA, USA: IEEE Computer Society Press.

Larsen, K. G., Petterson, P., & Yi, W. (1995). Model-Checking for Real-Time Systems. In H. Reichel (Hrsg.), *Proceedings of the 10th International Conference on Fundamentals of Computation Theory (FCT '95)* (S. 62-88). Berlin, Heidelberg: Springer.

Larsen, K. G., Petterson, P., & Yi, W. (19. Dezember 1997). *UPPAAL in a Nutshell.* Abgerufen am 31. März 2011 von http://www.it.uu.se/research/group/darts/papers/texts/lpw-sttt97.pdf

NTG. (1982). Zuverlässigkeitsbegriffe im Hinblick auf komplexe Software und Hardware. *Nachrichtentechnische Zeitschrift, 35*(5), S. 325-233.

Peterson, W. W., & Brown, D. T. (1961). Cyclic Codes for Error Detection. *Proceedings of the IRE, 49*(1), S. 228-235.

Petroff, B. (2009). Implementierung eines Beispielsystems zum Konzept der Entfernten Redundanz. *(Studienprojekt).* Universität Duisburg-Essen.

Sauler, J., & Kriso, S. (3. Dezember 2009). *ISO 26262 - Die zukünftige Norm zur funktionalen Sicherheit von Straßenfahrzeugen.* Abgerufen am 31. März 2011 von ELEKTRONIKPRAXIS: http://www.elektronikpraxis.vogel.de/themen/elektronikmanagement/projektqualitaetsmanagement/articles/242243/

Schneeweiss, W. (1999). *Die Fehlerbaum-Methode (aus dem Themenkreis Zuverlässigkeits- und Sicherheits-Technik).* Hagen: LiLoLe-Verlag.

Schwarz, J. (2005). *Risikoanalyse - Verfahrensbeschreibung.* Frankfurt: FAKRA-Entwurf.

Siewiorek, D. P., & Swarz, R. S. (1982). *The theory and practice of reliable system design.* Bedford, MA, USA: Digital Press.

Sneed, H. M. (1987). *Software-Management.* Köln: Verlagsgesellschaft Rudolf Müller.

Steinbeis GmbH & Co. KG für Technologietransfer, TransferZentrum Mikroelektronik. (8. September 2004). Beispielprogramm zur FlexRay-Kommunikation / FlexEntry USB-Stick.

The MathWorks. (18. Juni 2002). Online-Hilfe des Programms MATLAB.

U.S. Department of Defense. (1991). *Military Handbook: Reliability Prediction of Electronic Equipment.* Washington, D.C., USA.

Walter, M., & Schneeweiss, W. (2005). *The Modeling World of Reliability / Safety Engineering (Coosing the Best Method and the Best Tool for Quantitative Analysis).* Hagen: LiLoLe-Verlag.

Wenzel, T., Fassl, M., & Kalmbach, J. (11 2010). Rezept für sichere Software: Entwicklung von Steuergeräte-Basis-Software nach ISO/DIS 26262. *Elektronik automotive,* S. 52-55.

Wind River Systems. (4. Juni 2008). *ARINC 653: An Avionics Standard for Safe, Partitioned Systems.* Abgerufen am 31. März 2011 von http://www.computersociety.it/wp-content/uploads/2008/08/ieee-cc-arinc653_final.pdf

Zehbold, C. (2001). *Lebenszykluskostenrechnung.* Wiesbaden: Gabler.

Anhang A: UPPAAL-Deklarationen

```
const int minR = -7;            // Minimalwert für die Regeldifferenz
const int maxR = 7;             // Maximalwert für die Regeldifferenz

const int diffR = 4;            // Regeldifferenz einer künstlich hervorgerufenen Auslenkung
const int DiffD = 180;          // Zeitdauer, nach der die durch künstliche Auslenkung hervorgerufene
                                // Regeldifferenz auftritt

const int ZyklusD = 30;         // Dauer eines Regelzyklus
const int ReglerD = 15;         // Dauer einer Berechnung durch "RechnerRegler"
const int PruefD = 11;          // Dauer der Berechung des Passivierungskommandos

const int MotorD = 5;           // Dauer, die der träge Motor benötigt, bis er Kraft auf den
                                // Schlitten ausübt
const int SchlittenD = 20;      // Dauer, die der träge Schlitten benötigt, bis er sich bewegt


/* Zeittabelle eines Zyklus ("Zeitachse" nach unten):
  0       Beginn eines Zyklus
  0 bis  15  Berechnung der Regelfunktion
 15 bis  20  Verzögerung bis Motor ggf. Kraft ausübt, falls zuvor Stillstand
 15 bis  25  Verzögerung bis Motor ggf. Kraft ausübt, falls zuvor Drehung in Gegenrichtung
 26       Prüfung, ob ein Motor richtig angesteuert wurde
 20 bis  40  Verzögerung, bis sich der Schlitten ggf. bewegt, falls zuvor Stillstand
 20 bis  60  Verzög., bis sich der Schlitten ggf. bewegt, falls zuvor Bewegung in Gegenrichtung
 30       Ende eines Zyklus
*/

broadcast chan ReglerK [4];     // von "RechnerRegler[i]" an "Endstufe[i]" und "Passivierer[i][.]"
int [-1, 1] ReglerW [4] =       // Reglersignal[i]: -1 = vermindern, 0 = belassen, 1 = erhöhen
   {0, 0, 0, 0};                // mit Stillstand initialisiert, Index 0 unbenutzt

broadcast chan PassK [4][3];    // von "RechnerPassivierer[i,j]" an "Endstufe[j]"
int [0, 1] PassW [4][3] =       // Passivierung[i,j], 1 = Energie (keine Pass.), 0 = keine Energie
   {{0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}};     // Anfänglich keine Passivierung, Index 0 unbenutzt

broadcast chan EndstufeK [3];   // von "Endstufe[i]" an "Motor[i]".
int [-1, 1] EndstufeW [3] =     // Ansteuerung[i]: -1 = vermindern, 0 = belassen, 1 = erhöhen.
   {0, 0, 0};                   // Mit Stillstand initialisiert, Index 0 unbenutzt.

broadcast chan MotorK [3];      // von "Motor[i]" an "Schlitten".
int [-1, 1] MotorW [3] =        // Ausgeübte Kraft. -1 = vermindern, 0 = belassen, 1 = erhöhen.
   {0, 0, 0};                   // Mit "kraftlos" initialisiert, Index 0 unbenutzt.

int [minR, maxR] SchlittenW =   // Position relativ zum Sollwert. 0 = Gleichheit.
   0;                           // Anfänglich keine Abweichung.

broadcast chan TickerK;         // von "Uebereinstimmung" an "RechnerRegler".

int [1, 3] EINS = 1;            // Eins (für Parametrisierung von Prozessen erforderlich)
int [1, 3] ZWEI = 2;            // Zwei (für Parametrisierung von Prozessen erforderlich)
int [1, 3] DREI = 3;            // Drei (für Parametrisierung von Prozessen erforderlich)

int Zyklus = 0;                 // Zählt die Zyklen jeweils ab der künstlich hervorgerufenen
                                // Regeldifferenz mit 0 beginnend. Diese Variable wird nur für die
                                // Anfragen an den Erreichbarkeitsgraphen benötigt.

broadcast chan SensorK;         // von "Schlitten" an "SensorPos[.]"
```

```
chan RechSensK;                  // von "SensorPos[.]" an "RechnerSens"

int [minR, maxR] SollW = 0;      // Sollvorgabe (anfänglich 0)
int [-1, 1] RotW[3];             // Werte der beiden Rotationssensoren (Index 0 unbenutzt)
int [0, 1] RotS[3] =             // Signaturen der beiden Rotationssensoren (Index 0 unbenutzt)
   {0, 1, 1};
int [minR, maxR] PosW[4];        // Werte der beiden Positionssensoren und des virtuellen Positionssensors (Index 0
unbenutzt)
int [0, 1] PosS[3];              // Signaturen der beiden Positionssensoren (Index 0 unbenutzt)

int [0, 1] PassS [4][3];         // Signatur von Passivierung[i,j] (Index 0 unbenutzt)
```

Anhang B: VHDL-Testbench

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS
  -- Testbench zur Prüfung des Signaturverfahrens: Fortlaufend werden zufällige
  -- Nutzdaten erzeugt, senderseitig mit einer Signatur versehen und bei der
  -- simulierten Datenübertragung mit Fehlern behaftet. Es wird geprüft, ob der
  -- über die empfangenen Daten durchgeführte Signaturtest des Emfpängers den Fehler
  -- erkennt.

  COMPONENT sigma IS PORT ( -- Signaturerzeugung
        clk:    IN std_logic;
        i_data: IN std_logic_vector(20 DOWNTO 1);
        o_frame: OUT std_logic_vector(32 DOWNTO 1)
    );
    END COMPONENT;

    COMPONENT ueb IS PORT ( -- Datenübertragung unter Fehlerinjektion
        clk: IN std_logic;
        i_frame: IN std_logic_vector (32 DOWNTO 1);
        i_error: IN std_logic_vector (32 DOWNTO 1);
        o_frame: OUT std_logic_vector (32 DOWNTO 1)
    );
    END COMPONENT;

    COMPONENT tau IS PORT ( -- Signaturprüfung
        clk: IN std_logic;
        i_frame: IN std_logic_vector (32 DOWNTO 1);
        o_good : OUT std_logic
    );
    END COMPONENT;

    CONSTANT PERIOD: time := 10 ms; -- Taktung

    SIGNAL W_CLK: std_logic := '1';
    SIGNAL W_NEXTDATA: std_logic_vector(20 DOWNTO 1) := "11110000111100001111";
    SIGNAL W_FRAME_S: std_logic_vector(32 DOWNTO 1); -- gesendeter Frame
    SIGNAL W_ERROR: std_logic_vector(32 DOWNTO 1); -- injizierter Fehler (Bitmuster)
    SIGNAL W_ERRO1: std_logic_vector(32 DOWNTO 1); -- Fehler aus vorigem Takt
    SIGNAL W_ERRO2: std_logic_vector(32 DOWNTO 1); -- Fehler aus vorvorigem Takt
    SIGNAL W_FRAME_R: std_logic_vector(32 DOWNTO 1); -- empfangener Frame
    SIGNAL W_GOOD: std_logic; -- empfangener Frame als gültig bewertet

BEGIN
  -- Instanziierung von Sender, Empfänger und Fehlerinjektor
    se: sigma PORT MAP(clk => W_CLK, i_data => W_NEXTDATA, o_frame => W_FRAME_S);
    fi: ueb PORT MAP(clk => W_CLK, i_frame => W_FRAME_S, i_error => W_ERROR, o_frame =>
 W_FRAME_R);
    rec: tau PORT MAP(clk => W_CLK, i_frame => W_FRAME_R, o_good => W_GOOD);

  -- Takt
    W_CLK <= NOT W_CLK AFTER PERIOD/2;
```

```
    PROCESS(W_CLK)
            VARIABLE seed_a, seed_b: positive;
            VARIABLE rand1, rand2, rand3: real;
            VARIABLE int_rand1, int_rand2, int_rand3: integer;
    BEGIN
        IF(W_CLK'event AND W_CLK='1') THEN
          -- 12 Bit Bündelfehler erzeugen
            UNIFORM(seed_a, seed_b, rand1);
            int_rand1 := INTEGER(TRUNC(rand1*4096.0));

            -- maximal 20 Stellen nach links schieben
            UNIFORM(seed_a, seed_b, rand2);
            int_rand2 := INTEGER(TRUNC(rand2*20.0));

            -- zufällige Nutzdaten, 20 Bit
            UNIFORM(seed_a, seed_b, rand3);
            int_rand3 := INTEGER(TRUNC(rand2*1048576.0));

      -- Fehler injizieren und zwei Takte lang speichern
            W_ERROR <= std_logic_vector(SHIFT_LEFT(to_unsigned(int_rand1, 32),int_rand2
));
            W_ERRO1 <= W_ERROR;
            W_ERRO2 <= W_ERRO1;

      -- neue Nutzdaten in die Schaltung einspeisen
            W_NEXTDATA <= std_logic_vector(to_unsigned(int_rand3, 20));
        END IF;

      -- Warnung erzeugen, falls ein injizierter "tatsächlicher" Fehler
      -- (durch den mindestens ein Bit gekippt wurde) vom Empfänger nicht
      -- erkannt wurde (W_GOOD = '0'). Die Schaltung benötigt 30 ms (3 Takte),
      -- bis die ersten Nutzdaten beim Empfänger anliegen.
    ASSERT (W_GOOD = '0' OR to_integer(unsigned(W_ERRO2)) = 0 OR NOW < 30 ms)
        REPORT "UNDETECTED ERROR"
        SEVERITY warning;
    END PROCESS;
END behavior;
```

Anhang C: Nios II-Demoanwendung

```
#include <system.h>
#include <basic_io.h>
#include <altera_avalon_pio_regs.h>

#define LCD_CLR "[2J"                    // CLR-String
static unsigned char esc = 0x1b;         // ESC-Character

int main() {
    // LCD-Display öffnen, Gerätename -> system.h
    FILE* lcd_fd;
    lcd_fd = fopen("/dev/lcd_0", "w");
    if (lcd_fd == NULL) {
        printf("Unable to open lcd display\n");
    }

    // Begrüßungsmeldung anzeigen
    fprintf(lcd_fd, "%c%sRemote-Sig Demo\nv1.1 2011-02-23", esc, LCD_CLR);
    msleep(2000);

    // senderseitig vorliegende Daten
    alt_u32 indata = 0x0, indata_old = 0x0, comp_crc = 0xA1E, comp_sig =
0x2C2;
    alt_u8 seqnr = 0x0;

    // empängerseitig vorliegende Daten
    alt_u32 rec_data = 0x0, rec_crc = 0x0, rec_sig = 0x0, comp_crc_rec = 0x0;
    alt_u8 rec_seq = 0x0, exp_seq = 0x0;

    // vom Sender gespeicherte veraltete Daten
    alt_u32 indata_mem = 0x0, comp_crc_mem = 0x0, comp_sig_mem = 0x0,
seqnr_mem = 0x0;

    // Fehlerinjektor
    unsigned int fi = 0x0;

    // Diagnoseinformationen
    int seq_ok = 0x0, crc_ok = 0x0, sig_ok = 0x0;

    while (1) {
        // letzte Eingabe speichern
        indata_old = indata;

        // Eingabedaten den Schalterpositionen 11..0 entnehmen
        indata = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE) & 0xFFF;

        // neue Eingabe liegt vor
        if (indata != indata_old) {

            // Speicherung alter Daten für Fehlerinjektion
            indata_mem = indata_old;
            comp_crc_mem = comp_crc;
            comp_sig_mem = comp_sig;
            seqnr_mem = seqnr;

            // Sequenznummer und erwartete Sequenznummer erhöhen
            *((alt_u32 *) COUNTER_AV_0_BASE) = seqnr;
            seqnr =  *((alt_u32 *) COUNTER_AV_0_BASE);

            *((alt_u32 *) COUNTER_AV_0_BASE) = exp_seq;
            exp_seq =  *((alt_u32 *) COUNTER_AV_0_BASE);

            // CRC über Konkatenation aus Sequenznummer und Eingabedaten
berechnen
            *((alt_u32 *) CRC_AV_0_BASE) = (seqnr << 12) | indata;
            comp_crc = *((alt_u32 *) CRC_AV_0_BASE);
```

```
            // berechneten CRC signieren
            *((alt_u32 *) SIGMA_AV_0_BASE) = comp_crc;
            comp_sig = *((alt_u32 *) SIGMA_AV_0_BASE);
        }


        // "Datenübertragung" von Nutzdaten, Sequenznummer, CRC und Signatur
        rec_data = indata;
        rec_crc = comp_crc;
        rec_seq = seqnr;
        rec_sig = comp_sig;


        // Fehlerinjektion gemäß den Schalterpositionen 17..15
        fi = (IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE) & 0x38000) >> 15;
        switch (fi) {
            // nur Schalter 17: Eingabedaten invertieren. CRC und Signatur
werden als
            // ungültig erkannt.
            case 4: rec_data = rec_data ^ 0xFFF;
                    break;
            // Schalter 17 und 16: Eingabedaten invertieren und passenden CRC
berechnen. Eine
            // dazu passende Signatur kann nicht gefälscht werden.
            case 6: rec_data = rec_data ^ 0xFFF;
                    *((alt_u32 *) CRC_AV_0_BASE) = (seqnr << 12) | rec_data;
                    rec_crc = *((alt_u32 *) CRC_AV_0_BASE);
                    break;
            // Schalter 17, 16 und 15: Senden veralteter Daten. Zwar stimmen
CRC und Signatur,
            // die veraltete Sequenznummer wird jedoch erkannt.
            case 7: rec_data = indata_mem;
                    rec_crc = comp_crc_mem;
                    rec_sig = comp_sig_mem;
                    rec_seq = seqnr_mem;
                    break;
            default: break;
        }


        // LEDs entsprechend Eingabedaten und FI ansteuern
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, indata | fi << 15);

        // sender- und empfängerseitig vorliegende Daten auf dem LCD-Display
ausgeben:
        // jeweils: "<Sequenznummer> <Nutzdaten> (<CRC>|<SIG>)"
        fprintf(lcd_fd, "%c%s%2X %3X (%3X|%3X)\n%2X %3X (%3X|%3X)", esc,
LCD_CLR, seqnr, indata,
             comp_crc, comp_sig, rec_seq, rec_data, rec_crc, rec_sig);


        // Fehlerdiagnose Teil 1: CRC prüfen
        // ---------------------------------
        // CRC über empfangene Daten berechnen
        *((alt_u32 *) CRC_AV_0_BASE) = (rec_seq << 12) | rec_data;
        comp_crc_rec = *((alt_u32 *) CRC_AV_0_BASE);

        // berechneten CRC mit empfangenen CRC vergleichen
        crc_ok = (rec_crc == comp_crc_rec);

        // Fehlerdiagnose Teil 2: Signatur prüfen
        // --------------------------------------
        // prüfen, ob empfangene Signatur zu berechnetem CRC passt
        *((alt_u32 *) TAU_AV_0_BASE) = ((comp_crc_rec << 20) | rec_sig) &
```

```
0xFFF00FFF;
        sig_ok = *((alt_u32 *) TAU_AV_0_BASE);

        // Fehlerdiagnose Teil 3: Sequenznummer prüfen
        // -------------------------------------------
        // erwartete und empfangene Sequenznummer vergleichen
        seq_ok = (exp_seq == rec_seq);


        // Ausgabe der Fehlerdiagnose auf 7-Segment-Anzeige:
        // FA|CO <SeqOK> <CRCOK><SigOK>
        alt_u32 diag = sig_ok + (crc_ok * 0x100) + (seq_ok * 0x10000);
        diag += (sig_ok*seq_ok*crc_ok>0 ? 0xC0 : 0xFA) * 0x1000000;

        seg7_show(SEG7_LUT_8_0_BASE, diag);


        // "Flackern" des Displays verhindern
        msleep(100);
    }
    return 0;
}
```

Anhang D: Flexray-Kommunikation

```
#include <IO_S12DT256.h>
#include "FR_CC_MFR4300.h"
#include "global.h"
#include "FR_PL.h"
#include "sci.h"
#include "spi.h"
#include "dio.h"

void WAIT_TILL_CC_SYNC(void);
void FR_Communication(void);
void __premain (void);
int main(void);

extern ttMFR4300 *pMFR4300;

/*****************************************************************
*****************************************************************/
void WAIT_TILL_CC_SYNC(void) {
    u08 u08_Count = 0;

    ENABLE_CC_ACCESS;

    // Set CC to ready mode
    SET_POCCMD_MFR4300(MFR4300_CMD_READY);

    SET_POCCMD_MFR4300(MFR4300_CMD_ALLOW_COLDSTART);
    SET_POCCMD_MFR4300(MFR4300_CMD_RUN);

    while (GET_POCSTATE_MFR4300() != MFR4300_STATE_NORMAL_ACTIVE) {

        if (u08_Count++ <= 10) {
            DOUT_PT1 ^= 1;              //Toggle LED2
            ms_Delay(500);
        }
        else {
            SET_POCCMD_MFR4300(MFR4300_CMD_READY);

            SET_POCCMD_MFR4300(MFR4300_CMD_ALLOW_COLDSTART);
            SET_POCCMD_MFR4300(MFR4300_CMD_RUN);
            u08_Count = 0;
        }
    }
    DOUT_PT1 = 1;                       //set LED2

    printf_sci("FlexRay is synchronized\n\n");

    return;
}

/*****************************************************************
*****************************************************************/
void FR_Communication(void) {
    s08 s08_Function_Return;

    FRMSG FR_FlexMSG_TX1;

    u16 flexdata0 = 0; // Data sent/received on FlexRay
```

```
u16 flexdata1 = 0;

#ifdef NODE_0                                          //Variables for SPI communication

    u08 spidata; // Data received on SPI
    u08 received = 0; // Counts successful receive-operations on SPI

#endif

#ifdef NODE_1                                          //Variables for SPI communication

    u08 spidata0; // Data sent on SPI
    u08 spidata1;
    u08 spidata2;
    u08 spidata3;
    u16 delaycounter = 0;
    u08 sent = 0; // Counts successful send-operations on SPI

#endif

#ifdef NODE_0                                          //Initialisation of the FlexRay messages

    FRMSG FR_FlexMSG_TX2;

    FR_FlexMSG_TX1.u16_FrameID = 1;
    FR_FlexMSG_TX1.u16_Channel = MFR4300_CHA_B; //Data are sent on Channel A and B
    FR_FlexMSG_TX1.u08_Payload = 10;
    FR_FlexMSG_TX1.u08_PPI = 0;
    FR_FlexMSG_TX1.u16_Data[0] = 0;
    FR_FlexMSG_TX1.u16_Data[1] = 0;

    FR_FlexMSG_TX2.u16_FrameID = 5;
    FR_FlexMSG_TX2.u16_Channel = MFR4300_CHA_B;
    FR_FlexMSG_TX2.u08_Payload = 10;
    FR_FlexMSG_TX2.u08_PPI = 0;
    FR_FlexMSG_TX2.u16_Data[0] = 0;
    FR_FlexMSG_TX2.u16_Data[1] = 0;

#endif

#ifdef NODE_1

    FRMSG FR_FlexMSG_RX1_A;
    FRMSG FR_FlexMSG_RX1_B;

    FR_FlexMSG_TX1.u16_FrameID = 10;
    FR_FlexMSG_TX1.u16_Channel = MFR4300_CHA_B;
    FR_FlexMSG_TX1.u08_Payload = 10;
    FR_FlexMSG_TX1.u08_PPI = 0;
    FR_FlexMSG_TX1.u16_Data[0] = 0;
    FR_FlexMSG_TX1.u16_Data[1] = 0;

    FR_FlexMSG_RX1_A.u16_FrameID = 5;
    FR_FlexMSG_RX1_A.u16_Channel = MFR4300_CHA;      //Receive Buffer for Channel A
    FR_FlexMSG_RX1_A.u08_Payload = 10;
    FR_FlexMSG_RX1_A.u08_PPI = 0;
    FR_FlexMSG_RX1_A.u16_Data[0] = 0;
```

```
        FR_FlexMSG_RX1_A.u16_Data[1] = 0;

        FR_FlexMSG_RX1_B.u16_FrameID = 5;
        FR_FlexMSG_RX1_B.u16_Channel = MFR4300_CHB;      //Receive Buffer for Channel B
        FR_FlexMSG_RX1_B.u08_Payload = 10;
        FR_FlexMSG_RX1_B.u08_PPI = 0;
        FR_FlexMSG_RX1_B.u16_Data[0] = 0;
        FR_FlexMSG_RX1_B.u16_Data[1] = 0;

    #endif
                                                //init the SPI interface for accessing the FlexTiny modules
    initSPI(SPI1, SPI_BAUD_250kHz, SPI_MASTER, SPI_CPHA_RISE);

    Print_FT_Info(PL_CHANNEL_A);                //Print FlexTiny Information to the serial Interface
    Print_FT_Info(PL_CHANNEL_B);

    printf_sci("Enable Bus Drivers\n");         //Set Bus drivers to Normal Mode
    set_PL_MODE(PL_TJA1080, PL_CHANNEL_A, PL_MODE_NORMAL);
    set_PL_MODE(PL_TJA1080, PL_CHANNEL_B, PL_MODE_NORMAL);

                                                //Configure the Communication Controller
    if (INIT_CC_MFR4300() == FR_RET_SUCCESS) {
        DOUT_PT0 = 1;                           //Set LED0
        printf_sci("CC Configuration OK\n");    //Print Status of Configuration on serial Interface
    }
    else {
        DOUT_PT0 = 0;                           //Clear LED0
        printf_sci("CC Configuration NOT OK\n");
    }

    // Get Buffer Number that belongs to FlexRayID
    if (GET_BUFFERNUMBER_MFR4300(&FR_FlexMSG_TX1) != FR_RET_SUCCESS)printf_sci("TX1
Buffer Error\n") ;

    #ifdef NODE_0
        if (GET_BUFFERNUMBER_MFR4300(&FR_FlexMSG_TX2) != FR_RET_SUCCESS)printf_sci(
"TX2 Buffer Error\n") ;
    #endif

    #ifdef NODE_1
        if (GET_BUFFERNUMBER_MFR4300(&FR_FlexMSG_RX1_A) != FR_RET_SUCCESS)printf_sci(
"RX1 A Buffer Error\n") ;
        if (GET_BUFFERNUMBER_MFR4300(&FR_FlexMSG_RX1_B) != FR_RET_SUCCESS)printf_sci(
"RX1 B Buffer Error\n") ;
    #endif

    WAIT_TILL_CC_SYNC();                        //wait until FlexRay is synchronized

    #ifdef NODE_0
    initSPI(SPI0, 0xFF, SPI_SLAVE, SPI_CPHA_RISE);
    #endif

    #ifdef NODE_1
    initSPI(SPI0, 0xFF, SPI_MASTER, SPI_CPHA_RISE);
    #endif

    while(1) {
```

```
        // Equal Code for Node0 and Node1

        s08_Function_Return = PUT_FLEXRAYMSG_MFR4300(&FR_FlexMSG_TX1);              //Try
to send FlexRay Message
        if (s08_Function_Return == FR_RET_SUCCESS) {
//Message was successfully sent

            if (FR_FlexMSG_TX1.u16_Data[0] < 50000) {
//Increase Data
                FR_FlexMSG_TX1.u16_Data[0] += 100;
//Both Nodes send data Ramp
            }
//from 0 to 50000 on their
            else {
//SyncFrame (Node0 ID1; Node1 ID10)
                FR_FlexMSG_TX1.u16_Data[0] = 0;
            }
        }
        else if (s08_Function_Return == FR_RET_NOSYNC) {
//Synchronisation Lost
            DOUT_PT1 = 0;
//clear LED1
            printf_sci("FlexRay lost synchronization\n");
            WAIT_TILL_CC_SYNC();
        }


        // Different Code for Node0 and Node1
        // Node1 receives Data on FlexRay and transmits the Data on SPI
        // Node0 receives Data on SPI and transmits the Data on FlexRay

        #ifdef NODE_0

            if (ReceiveSpiData(SPI0, &spidata) == 0) {
                printf_sci("Empfangen: %x\n", spidata);
                received++;

                if (received <= 2) {
                    flexdata0 = (flexdata0 << 8) | spidata;
                }
                else {
                    flexdata1 = (flexdata1 << 8) | spidata;
                }
            }

            if (received == 4) {
                received = 0;
                FR_FlexMSG_TX2.u16_Data[0] = flexdata0;
                FR_FlexMSG_TX2.u16_Data[1] = flexdata1;
                PUT_FLEXRAYMSG_MFR4300(&FR_FlexMSG_TX2);     //Try to send FlexRay Message
            }

        #endif

        #ifdef NODE_1

            s08_Function_Return = GET_FLEXRAYMSG_MFR4300(&FR_FlexMSG_RX1_A);
```

```
//Check if new Data are available on CH A
            if (s08_Function_Return == FR_RET_SUCCESS) {                                    //If
new Data were received
                flexdata0 = FR_FlexMSG_RX1_A.u16_Data[0];
                flexdata1 = FR_FlexMSG_RX1_A.u16_Data[1];

                spidata1 = flexdata0 & 0xFF;
                spidata0 = (flexdata0 >> 8) & 0xFF;
                spidata3 = flexdata1 & 0xFF;
                spidata2 = (flexdata1 >> 8) & 0xFF;

                delaycounter = 0;
                sent = 0;
                while (sent == 0) {
                    if (delaycounter == 0) {
                        TransmitSpiData(SPI0, spidata0);
                        printf_sci("Gesendet 1/4: %x\n", spidata0);
                    }
                    else if (delaycounter == 200) {
                        TransmitSpiData(SPI0, spidata1);
                        printf_sci("Gesendet 2/4: %x\n", spidata1);
                    }
                    else if (delaycounter == 400) {
                        TransmitSpiData(SPI0, spidata2);
                        printf_sci("Gesendet 3/4: %x\n", spidata2);
                    }
                    else if (delaycounter == 600) {
                        TransmitSpiData(SPI0, spidata3);
                        printf_sci("Gesendet 4/4: %x\n", spidata3);
                    }
                    else if (delaycounter == 800) {
                        sent = 1;
                        delaycounter = 0;
                    }
                    delaycounter++;
                }
            }

        #endif
    }

    return;
}


/*****************************************************************
*****************************************************************/
void __premain (void) {
}


/*****************************************************************
*****************************************************************/
int main (void) {
    pMFR4300 = (ttMFR4300*)MFR4300_BASE_MEM_OFFSET; //cast global FlexRay structure to memory before
                                                    //Interrupt is enabled
```

```
    initMCU();                                          //Initialize the STAR12 µC
                                                        //Enable Interrupts
    DIR_PT = 0xFF;                                      //Initialize the PORTT Direction as Output
    DOUT_PT = 0x00;                                     //Switch Off all LEDs

    initSCI(SCI0, SCI_BAUD19200, MODE_8N1, TX_ON, RX_ON, SCILOOP_OFF, WAIT_DIS);
                                                        //Initialize the SCI0

    #ifdef NODE_0
        printf_sci(" NODE_0\n\n");
    #endif
    #ifdef NODE_1
        printf_sci(" NODE_1\n\n");
    #endif

    *(u16 *)(0x4000) = 0x0;                             //After Reset the first bus access is not
                                                        //executed => use dummy access

    FR_Communication();                                 //Start FlexRay Communication

    while(1) {}
}
```

Dateigerüst und einzelne Programmabschnitte übern. aus
(Steinbeis GmbH & Co. KG für Technologietransfer, TransferZentrum Mikroelektronik, 2004)